Conceptual modeling and the Lexicon

Hoppenbrouwers, J.J.A.C.

Publication date:
1997

Document Version
Publisher's PDF, also known as Version of record

CentER
for
Economic Research

Conceptual Modeling and the Lexicon

Jeroen Hoppenbrouwers

Tilburg University

CONCEPTUAL MODELING AND THE LEXICON

THESIS PUBLISHERS
AMSTERDAM 1997

CONCEPTUAL MODELING AND THE LEXICON

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Katholieke Universiteit Brabant, op gezag van de rector magnificus, prof. dr. L.F.W. de Klerk, in het openbaar te verdedigen ten overstaan van een door het college van decanen aangewezen commissie in de aula van de Universiteit op vrijdag 17 oktober 1997 om 11:15 uur

door

Jeroen Johannes Antonius Catharina Hoppenbrouwers

geboren op 22 juli 1967 te Bergeyk.

Promotor: prof. dr. R.A. Meersman
Copromotor: dr. H. Weigand

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems, and CentER, the Graduate School of the Faculty of Economics and Business Administration of Tilburg University.

ISBN 90-5668-027-7

Voor mijn ouders

Contents

Preface

When I was looking for a PhD project back in 1990, Hans Weigand from Tilburg University had just started looking for a PhD student to do work on language-related aspects of Conceptual Modeling. My background in computer science and applied (psycho)linguistics acquired at Eindhoven University of Technology, under supervision of Jan Ulijn, had prepared me for the job, and in March 1991 I started work on the role of the Lexicon in conceptual modeling, as part of the LIKE project.

Initially, my task was to investigate tools in the form of work benches to support the requirements phase and the design and analysis phase of a project on the basis of linguistics and object orientation. Both theoretical structures, mostly expressed in Simon Dik's Functional Grammar, and tools such as computerized dictionaries and thesauri were to be examined and applied to conceptual modeling problems. Later on, preliminary design of such a computerized Lexicon was added to the project.

My promoter Robert Meersman quickly discovered my tendency to never say 'no' to a technical challenge, and soon I became fully immersed in the Faculty's Infolab, often working more on the Lab's infrastructure and other interesting aspects of modern information technology than on my research. But both Robert and Hans once in a while kicked me back to my corner to get something done, and in the end, they succeeded in getting me to do some thesis-related work.

A research group consisting of Bram van der Vos, Hans Burg, Ans Steuten, Victor van Reijswoud, Nardo van der Rijst, and Egon Verharen, supervised by Reind van de Riet, Jan Dietz, and Hans Weigand, offered the opportunity to exchange views and opinions on the subject. I learned that even within the small scientific field of conceptual modeling, the number of different approaches was staggering. So the next few years I spent exercising my fluency in various formal expression paradigms (from both linguistics and conceptual modeling), trying to find parallels between them, and eventually to come up with common properties related to terminology. Many intense discussions with Hans and the other people at the Infolab provided me with an ever-increasing stack of interesting ideas, but it turned out to be difficult to free myself from the traditional conceptual modeling paradigms that still looked at modeling from a database point of view.

During that time, my family had to endure strange monologues about abstract concepts, that were often partially common sense and partially pure mystique. But they listened nonetheless, and I am convinced that their patience in this has helped me a great deal to work out the many twists in my first approaches. Before driving to the university early in the morning, I usually had already discussed subtle details

with my father and refined them on the way. Inevitably, something else popped up during the day and I would go home in the evening with my head full of problems, ready to be poured out right over my parents as soon as I opened the door.

Things got even worse when my brother joined the Infolab to assist in the Grammalizer project. The weird situation of two brothers working together on the same subject and car pooling to the university provided us with the unique opportunity to have plenty of project meeting time each day. These discussions were especially valuable during the last stages of my PhD project, when despair and panic often raised their heads. They also drove our colleagues crazy, especially Willem-Jan van den Heuvel who regularly had to face fundamental overnight changes in almost everything.

Although the Grammalizer project, and later TREVI, significantly increased my workload and undoubtedly caused delay of the final thesis publication, both projects have contributed a great deal to the practical value of my work, or so I would like to believe. They provided the opportunity to actually implement many of our ideas and to test them in real-world situations. On top, the stimulating environment of the Infolab was never boring and always provided enough input for endless debate and technical tinkering—often more input than my PhD project could handle.

When finally the thesis materialized, the reading committee consisting of Reind van de Riet, Mike Papazoglou, Walter Daelemans, and Terry Halpin made sure that my ideas were sound and kindly suggested improvements in weak areas. Olga de Troyer, who supervised the Grammalizer and TREVI projects, allowed me to spend time on my thesis in between the rush for project deadlines. Frans Laurijssen is doing a great job actually coding my design in Java, while Ellen-Petra Kester and my brother Stijn struggle to specify English and Spanish lexicons in LIX. And when at some point everything seemed to come to a grinding halt, Alice Kloosterhuis always found a way through the maze of the University's dark alleys to get things done in time.

I wish to thank Hans Weigand, my family, and all other people from the Infolab for their support during the Thesis Years.

Jeroen Hoppenbrouwers
Valkenswaard, September 1997

Chapter 1

Introduction

— in which the author briefly indicates what the motivation for this work was and where the reader will get a fair overview of what can be expected —

One of the recurring problems in any medium or large scale development project, and especially in the development of information and communication systems, is the gap between the system requesters on one side and the system providers on the other. Many methods and approaches to narrow this gap have been proposed, and currently there is a rich choice of public and commercial methods available to help system designers to better serve their customers. But despite ample choice, complaints about information systems that do not live up to the buyer's expectations are heard more often than ever before.

Some of the problems may be caused by purely technical issues, such as an inadequate platform, bad software engineering, or grossly underestimated system loads. Sometimes the environment in which the system works changes so drastically that the system cannot keep up and has to perform tasks that it was never designed to perform in the first place.

But in many cases, the core design of the system, in the most abstract representation possible, is simply not appropriate for the environment in which it has to work, despite a great deal of communication between the people involved. Such a mismatch between the intentions of the buyers and the perceptions of the sellers is only explainable by assuming a big *communication problem.* It is this communication problem which I want to address.

1.1 Aim of this thesis

This thesis tries to contribute to the technical and scientific field of *conceptual modeling*, the art and science of analytical description of a well-defined part of the real world in such a way that a machine can eventually support some (usually administrative) aspects of that world. I will use theoretical ideas and empirical facts from various, often seemingly unrelated scientific fields, such as cognitive psychology, computer science, lexicography, psycholinguistics, software engineering, philosophy, and linguistics, to develop a central tool which should improve mutual communication between all the people who work on a new information system.

I start from the observation that people tend to center their communication efforts around *words* (Weigand, 1990), and that these words carry some *meaning*—they are not just plain senseless labels, such as identification codes. Words are the visible and audible elements of a huge system of knowledge, carried by both language and culture, in which people freely move around and in which context all linguistic utterances have to be interpreted. Philosophers such as Hamann, Wittgenstein, Heidegger, Foucault, Frege, and Peirce even state that words and language are the essence of both knowledge and human behavior (Weigand, 1990). Words might not be the central core of knowledge, but they certainly are of paramount importance in human communication—in knowledge *transfer*.

Since knowledge transfer is one of the main purposes of conceptual modeling, thoughtful, rigorous application of 'latent knowledge' that is 'hidden' in commonly used words should improve the usability of conceptual models, because people would be able to better 'connect' their mental model of the domain to the conceptual model. Moreover, using the 'correct' words in a model should drive the model towards a 'natural' representation of the domain. Therefore, my central research question is:

> "How can the implicit knowledge that is contained in terminology be successfully used to improve conceptual domain analyses?"

Because most conceptual modeling methods already use words, usually as labels to identify their conceptual elements, I will mostly focus on *improvements* in word choice and phrase type, e.g., replacing single prepositions by a complete verb phrase in a consistent manner.

Combining the theories and ideas of lexicographers, caretakers of words *par excellence*, with those of knowledge workers such as information scientists and logicians, I try to find a way to look at conceptual modeling from a terminology point of view. On the way, I encounter a number of existing methods of information analysis, dictionary construction, language analysis, and model building, that all emphasize different features of the world they try to capture. My goal is to carefully examine each of the viewpoints involved, and to bring together those aspects which allow me to connect it all in a satisfactory way.

Once I have settled on a set of essential aspects, I can design a software system, called the Lexicon Management System, which should make the discovered generic aspects of the previously mentioned world views accessible enough for them to be used in real-world conceptual modeling by information analysts who are trained in neither lexicography, nor linguistics. As an additional research question, or better design question, I therefore pose the following:

> "How can a Lexicon be designed, built, and used to effectively support Conceptual Modeling in real-world projects?"

Another important aspect in the usability of a Lexicon is its (technical) appearance. Many small, experimental systems have been built on stand-alone personal computers, using tools that were quite appropriate for the scientific project targets but not fit to support practical applications in a production environment. The Lexicon design as described in this thesis is aimed at large-scale production environments, providing multi-user access and high performance on an appropriate platform. It

can be integrated in external applications when used as a network server, and just as with traditional database systems, can be shielded from casual or non-technical/non-linguist users with appropriate front ends. In the appendices, I include the results of work that has been done on these technical aspects.

1.2 Software Development and Requirements Engineering

Luqi and Goguen (1997) write:

> "Experience shows that many of the most vexing problems in software development arise because any computer system is situated in a particular social context. Moreover, much of the information needed to design a system is embedded in the worlds of users and managers, and is extracted through interaction with these people. This information is informal and highly dependent on its social context for interpretation. On the other hand, we define the programming and other representations used to construct computer-based systems using formal syntactic and semantic rules. Both the formal, context-insensitive, and the informal, socially situated aspects of information are crucial for success."

They call these two aspects *the dry* and *the wet*, and their reconciliation has been claimed to be the essence of requirements engineering (Goguen, 1994). There are degrees of formalization, ranging from the very informal *wet*, typically found during feasibility study and requirements elicitation, to the very formal *dry*, usually encountered in the final analysis and design stages. The transition from *wet* to *dry* should be gradual, with good traceability between formalization degrees. There is no 'ideal' degree of formalization; in particular, as an aid to future modifications it is highly desirable to make contextual (*wet*) information available along with specifications and code. Clearly, it would be better if such information were systematically recorded in the first place (Luqi and Goguen, 1997).

Wet specifications are strongly related to people, communicating in a natural language such as English. The fact that people use natural language, widely assumed to be informal as opposed to e.g. logic (Gamut, 1991), does *not* mean that these people's work and ideas therefore must be informal as well. Natural language, if used properly, can very well convey formal notions (Weigand, 1990).

But there is even a direct theoretical link between linguistics, especially the field of semantics, and conceptual modeling:

> "The object of natural language semantics is to provide a knowledge representation formalization into which a parsed sentence can be mapped. We will never be able to claim the knowledge representation problem is solved until we have a reasonably complete knowledge representation system for natural language, which we do not have now."
>
> (Abbott, 1987)

In this thesis, I try to bring the worlds of semantics and conceptual modeling closer together. I do this not by focusing on a generic theoretical framework for semantic relationships or quantification, but by concentrating on a particular aspect of natural language, its *vocabulary*, or a bit less precise, its *words*.

Working with words in conceptual modeling has been preceded by massive efforts in word description for other technical and scientific fields. Bloomfield (1933) considered the Lexicon "just the collection of irregularities," but current insights in linguistics and natural language processing (NLP) tend to emphasize the role of the Lexicon more and more (Boguraev and Briscoe, 1989; Briscoe, 1991). The recent availability of cheap memory machines in the form of generic computation engines with large mass storage devices has enabled people to start gathering lexicographical material on an unprecedented scale (Calzolari, 1993a).

The structure of a proper Lexicon (as an artifact), the provider of the words people use to communicate in natural language, can be as formal as every other formal theory for domain modeling (Sowa, 1983; Boguraev and Briscoe, 1989). As such, a Lexicon can be of help during the information acquisition and the validation runs of the (partially complete) formal model of the system, alongside the other formal models currently in use (Hoppenbrouwers et al., 1996).

1.3 Linguistic Instruments in Knowledge Engineering

In the late 1980s, a small group of people at Amsterdam Free University started work on applications of linguistics to general problems of conceptual modeling. This resulted in two PhD theses (Weigand, 1990; Dignum, 1989).

The work of Weigand concentrated on a generic framework for knowledge base systems, strongly inspired by Functional Grammar (Dik, 1989). Weigand was one of the first to show a fundamental parallel between insights in theoretical linguistics (and logic) and the practice of conceptual modeling in information systems engineering; before his work, language was merely seen as a convenient heuristic to facilitate modeling in one of the specialized IT modeling paradigms.

Dignum took Functional Grammar, combined it with deontic and modal logic (among others), and turned the result into a coherent language called Conceptual Programming Language (CPL) in which many aspects of information systems can be expressed. Both the efforts of Weigand and Dignum therefore had their bases in linguistics and logic, and not in the 'traditional' CM fields of computer science and artificial intelligence.

A group of researchers (Jan Dietz, Simon Dik, Robert Meersman, Willem Meijs, Reind van de Riet, and Hans Weigand) subsequently set up a workshop—held in early 1991 in Tilburg (The Netherlands)—in which people from the disciplines of linguistics, AI, CS, CM, and Management Science shared their insights and views on Linguistic Instruments in Knowledge Engineering, or LIKE as it became known (van de Riet and Meersman, 1992). The resulting volume contains a diverse selection of papers, all related to conceptual modeling, and provided enough leads to extend the research group with some PhD students at the universities of Maastricht (later Delft), Amsterdam, and Tilburg, later known as the LIKE group.

Burg (1996) concentrated on reuse and simulation of conceptual models and subsequent code generation. His modeling language COOPR-$\mathcal{X}$ is an experimental implementation, both graphical and symbolical, of Dignum's CPL, with traces of MOKUM (van de Riet, 1989). Van Reijswoud (1996) took the DEMO modeling language (Dietz, 1990) that emphasizes the actual communication between agents in an environment, enriched it with several success and failure features, and applied the result to some real-world cases. Steuten (1996, 1997) worked on illocutionary expressions and used Functional Grammar to analyze business conversations. She concentrated on the internal structure of the separate expressions and the mutual dependencies between them. Verharen (1997) described both formal and practical ways to specify the behavior of intelligent agents in the context of Speech Act Theory, the Language-Action Perspective. Van der Vos did work on verification of conceptual models through linguistic analysis (Gulla et al., 1996; Hoppenbrouwers et al., 1996). Hoppenbrouwers' work centered around the Lexicon and is in front of the reader.

LIKE eventually caused the start of an ongoing series of international workshops under the umbrella name 'Natural Language and Databases.' The first three were held in Versailles (France), Amsterdam (the Netherlands), and Vancouver (Canada), and saw a steady growth in interest throughout the CM community as more and more practical applications of LIKE surfaced and industry got interested in what until then remained a theoretical exercise (Bouzeghoub and Métais, 1995; van de Riet et al., 1996; McFetridge, 1997).

An example of this industry attention is the Grammalizer project (Hoppenbrouwers et al., 1997a), aimed at a production-quality CASE tool to support KISS analysts with the initial sentence analysis and subsequent knowledge extraction and organization. Sponsored by both KISS BV and the Dutch Ministry of Economic Affairs, the Grammalizer is a framework that combines state-of-the-art techniques in client/server architecture, natural language processing, and computational lexicography to enable integration of NL sentence analysis in the current practice of conceptual modeling. For more information, see Appendix B.

The ESPRIT project TREVI, which aims at producing a toolkit for the assembly of customizable indexing and retrieval engines, contains a Lexicon module which very closely follows the design as presented in Chapter 4. The TREVI Lexicon is discussed in Appendix C.

1.4 Overview of this Thesis

In the first part, I discuss backgrounds and theoretical approaches to lexicons and conceptual modeling. Chapter 2 gives an overview of current practice in requirements engineering and the role of terminology in the process. From this, I derive the role a Lexicon may play in a CASE system, and give some examples of Lexicon usage in conceptual modeling. Chapter 3 reviews various issues related to natural language processing and Functional Grammar, leading to my proposal for the internal structure of a Lexicon. I have split up the structure in a Lexical Base, containing mainly orthographic and morphologic information, and a Concept Base, containing semantic information. Chapter 4 then is a detailed, formal description of

my Lexicon design following the structures of Chapter 3, fitted for implementation in a generic Lexicon Management System.

Part two of this thesis contains two example cases of possible Lexicon usage in two current conceptual modeling methods. I focus on the improvements that explicit usage of lexical knowledge could bring over more traditional approaches. Chapter 5 considers the lexical aspects of Object-Role Modeling, and proposes various rules and conventions to improve ORM's linguistic fundamentals. Chapter 6 discusses KISS, a method for conceptual modeling based on linguistic notions, and presents some approaches and extensions that draw heavily upon the Lexicon.

Chapter 7 contains a summary of the conclusions and gives ideas for future work. The first appendix lists features of some current, implemented Lexicons that have inspired my work and have provided a significant part of the lexical contents of our own Lexicon implementation. The subsequent appendices present an overview of the Grammalizer project and especially its Lexicon, and the Lexicon part of the TREVI project, both of which are based on the structures of Chapter 4.

Part I

Backgrounds and Theory

Chapter 2

Conceptual Models and the Lexicon

— in which the author discusses various ways of using a Lexicon as an aid in Conceptual Modeling, and gives some preliminary hints to practical applications of such a Lexicon —

The world has come a long way since the time that programmers had to work with a bare-bones compiler or assembler, a linker, and some low-level debugging tools (or worse). As the complexity of the produced software systems grew, so did the tools of the trade. For the bigger projects with hundreds of systems analysts, designers, and programmers (and technical writers, quality assurance people, beta testers, librarians, and secretaries, all essential people, but all too often forgotten), any progress would be unthinkable without large-scale systems to support the whole mutual effort. Add to this the incredible pace at which systems developers must work to keep in touch with their organizations' rapid restructuring and business process redesign projects, and the need for computerized support becomes paramount.

The systems to assist the whole process of software engineering are usually grouped together under the umbrella name of Computer-Aided Software Engineering (CASE) tools. They range from very abstract, analytical 'upper CASE' tools to concrete, yet advanced integrated development environments ('lower CASE'). Some approaches even try to skip the lower level tools altogether and *generate* (part of) a system directly out of high-level specifications. For a more complete discussion of CASE tools, see Burg (1996).

In software development, the following phases can be recognized: Analysis, Design, Implementation, Testing, Exploitation, and Maintenance (NGGO 1994). In this thesis I will mainly concentrate on the Analysis phase.

2.1 Requirements Engineering

Loucopoulos and Karakostas (1995) define Requirements Engineering as:

> "...the systematic process of developing requirements through an iterative, cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained."

Requirements Engineering covers the analysis phase, but also extends it phase with an initial *requirements elicitation* process. In this process, requirements of the users are acquired and combined with domain knowledge. This process is followed by *requirements specification* or *conceptual modeling*, where abstract models of the Universe of Discourse are created, guided by the domain knowledge and the user requirements. Subsequently these models are verified and validated in close cooperation with the users (Burg, 1996, p.44).

The NATURE project (Jarke et al., 1994) recognized three dimensions of Requirements Engineering: specification, representation, and agreement. It was assumed that at the beginning of each project, the requirements as projected on the three axes were opaque, informal, and personal, and that the aim of requirements engineering was to refine the requirements to be complete, formal, and representing the common view, respectively. Representation of the requirements did not concentrate on the eventual (formal) outcome alone; NATURE strongly advocated to use both informal, semi-formal, and formal representations where they fitted the best, and supported the transformation between them. Jackson *et. al.* (1995) describe such an integrated environment.

Due to the informal character of the first step of requirements engineering—the elicitation process—the resulting document usually is written in a natural language (NL) such as English. Some analysis methods such as KISS (Kristen, 1994) take such an NL document and use linguistically inspired heuristics to extract potentially important concepts directly out of the text. Limited computer support for this process is being developed in the Grammalizer Project (Hoppenbrouwers et al., 1997a).

After initial elicitation and specification, the resulting conceptual models should be verified, i.e., checked for internal consistency. Some inconsistencies can be prevented by the method itself, others have to be removed manually (Wijers and Heijers, 1990). Reducing or eliminating redundancy is another target of verification.

The last step in requirements engineering is validation, where the complete result of the work is presented to the users and certified for correctness against the users' intentions (El Emam et al., 1996). The following aspects should be subject to validation (Theodoulidis and Alexakis, 1993):

External consistency: agreement between what is stated in the requirements model and what is true in the problem domain.

Non-ambiguity: a requirement cannot be interpreted in more than one way.

Minimality: no over-specification, i.e., include nothing in the requirements model that is not necessary or simply not wanted.

Completeness: no omission of essential information about the problem domain.

There are various methods to validate a conceptual model (Burg, 1996, p.63), of which especially NL *paraphrasing* is of interest to me. By paraphrasing or verbalizing a formal, usually graphical conceptual model (Halpin and Harding, 1993), the user receives a transcription of the model in plain NL sentences, which should be readily understandable without any formal background. Aim of the paraphrasing process is to capture as much of the formal semantics as possible, which can lead to massive amounts of small, restricted language sentences if the conceptual modeling paradigm does not agree well with NL structures. Paradigms based on NL notions

therefore have the advantage here (Wu and Flynn, 1995). Sophisticated paraphrasing techniques do not produce separate sentences, but combine them into larger units (Gulla and Willumsen, 1993).

2.1.1 Goals of Requirements Engineering

Independent of the way in which each method for requirements engineering approaches the problem, their final aims are the same (Hofmann, 1993):

- To enforce users to consider their requirements carefully and to review them within the context of the problem.
- To get specifications of the problem, its domain, and its solution that are as correct and complete as possible.
- To bring both the users and the developers in agreement about what the requirements actually are.
- To act as a contractual agreement between users and developers against which the design and implementation results are checked for completeness and correctness.
- To provide a starting point for project management activities by giving estimates for cost, time, and required resources.
- To establish the users' commitment to the software development project and result, by letting them participate in this project in an early stage.

Wieringa (1996) also lists some desirable properties of a requirements specification:

Communicability: The specification should serve as a channel of communication about the system among all stake holders. It must be both understandable and unambiguous.

Validity: The specification should specify the requirements on the system accurately.

Implementation independence: No implementation decisions should be made during the process of creating the requirements specifications.

Completeness: The specification should be as complete as possible, given the available time and attainable agreement.

Feasibility: The specification should describe behavior that can actually be realized in a system and is cost-effective.

Consistency: Descriptions of a system should not conflict with each other.

Verifiability: It should be possible to observe whether a system satisfies its specification, which means that properties of the system should be specified in a measurable way.

Maintainability: Changes in requirements, also called *creeping requirements* (Anthes, 1994), both before and after the delivery of the system, should be easily incorporable in the specification. This also requires forward and backward traceability (Wieringa, 1995) to match requirements and design.

2.1.2 Terminology in CM

In many of the above approaches, aims, and discussed documents, NL plays an important role. Essential in any NL document, but also in many formal models, is the used *terminology*. Words transfer the bulk of meaning and content, whereas syntax, however indispensable it might be, is mainly a framework to put words in an accepted order (Beedham and Bloor, 1989). Elicitation pivots around word recognition. Specification is about word choice and agreement on word meaning. Verification considers relationships between words and their consistency with known parts of the world. Validation can only succeed if the words used to verbalize a conceptual model are recognized and correctly understood by the users.

To process all the documents mentioned above and to successfully reach the stated aims and intentions, terminology standardization and adherence to this terminology is of paramount importance. In this view, a Lexicon, containing both domain terminology and accepted common terminology, becomes an indispensable tool for requirements engineering, and because of the traceability requirement, for design and implementation as well.

2.1.3 Place of the Lexicon

CASE systems are typically built around one or several development environments, complemented with analysis and design aids which are in their turn based on one or several conceptual modeling paradigms. Most CASE tools nowadays have their own centralized repository as well. Such repositories store extensive development and design documentation, versioning information, source code change logs, diagrammatic schemes, database table descriptions, data dictionaries, and other indispensable system information. This collection of documents is in most cases stored in a coherent manner and guarded by some library management system that also enables indexing and retrieval, and sometimes provides facilities for extensive user queries and the production of management information.

The Lexicon is neither designed nor intended for any of these purposes. It is meant to facilitate and enhance the process of conceptual modeling itself, not the management of its results. It takes this role because the concepts in the domain that is to be modeled are not *created* during conceptual modeling, they are merely *discovered*. The Lexicon serves as an explicit store for these concepts and the linguistic 'handles' by which they are communicated, and offers pre-existing concepts that might be re-used or give hints about new concepts.

The Lexicon deserves a prominent place among the other tools that together make up an (upper) CASE system, but it does not replace any of them. Only the Data Dictionary might be a candidate for careful redesign, and even this module will still offer some specialized features[1] that the Lexicon was never designed to provide.

[1] The main difference between a Data Dictionary and a Lexicon is the fact that the former is a text, meant for human readers, while the latter is a database, meant for machine usage (Guthrie et al., 1996).

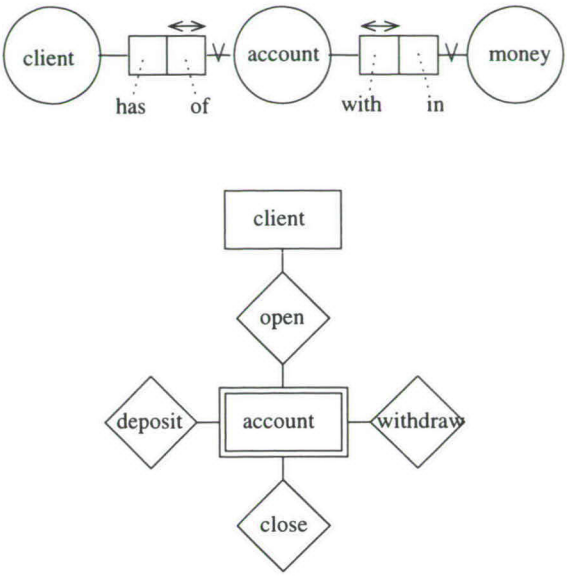

Figure 2.1: A NIAM and KISS representation of the same partial domain description

2.2 The Lexicon as Terminology Repository

Despite their sometimes quite unrelated approach, notation, and paradigm, most conceptual modeling methods share a common feature: they use words to identify and relate the concepts in the domain. Assuming that they all start with the same user input (elicitation), they will also adopt the same domain vocabulary. When the domain-specific words are left out of the resulting models, they might seem completely unrelated, but with the terminology included, overlaps and gaps become visible.

As an example, see Figure 2.1, which presents a NIAM (Nijssen and Halpin, 1989) and a KISS (Kristen, 1994) model which both stem from the same NL domain description:

> *After an account has been opened for the client, money may be deposited in the account, and be withdrawn from it by the client. Clients may open more than one account, and close them at will.*

Although both example models present a different view on the domain (NIAM a data-oriented view, KISS an action-oriented one), and while they do not use all of the available terminology, there clearly is overlap, and this is not a coincidence. The users' terminology is linked to concepts in their domain, and most modeling methods prefer to retain this terminology if they isolate such concepts. Therefore, when the same terminology surfaces in different models (of the same domain), it is likely that the models also contain the same concepts. Through terminology, models can be related to each other, and to the mental domain model that the users have developed for themselves. Ignoring this link, e.g., by replacing the users' terminology by

meaningless numerical identifiers, would theoretically retain the model's semantics, but in practice lead to severe communication problems. The terminology provides the connection between the modeling paradigm's abstract view on the world and the users' intuitions. As such, it is essential for the process of requirements engineering, and deserves great attention—as much attention as the formal capture of concepts and relations, the traditional mainstay of conceptual modeling efforts.

2.2.1 Language Registers

Another reason to increase CM's emphasis on terminology analysis can be found in empirical socio-linguistic research.

It has long been recognized that groups of people working together develop their own *register* of language (Ulijn and Strother, 1995). It has been found that the lower levels of language (orthography and morphology) are common between most users of a certain language, while lexis and even syntax[2] can vary considerably between groups of people. In this thesis I will not discuss the differences in register at the level of discourse and semantics, because those linguistic levels are too far above the lexical level of the Lexicon. Various socio-linguistic and psycholinguistic studies have shown that grammatical form is less important in communication than lexis (Beedham and Bloor, 1989), so I will also drop the syntactic aspects. What remains is the *vocabulary* part of a register.

Research into language registers has concentrated on finding macro variances in language use between e.g. scientists, technicians, business people and 'common' people, with the first using the most specialized register and the last the least specialized. But in the context of conceptual modeling, the micro variance (particularly in vocabulary) is the most interesting. Ulijn and Strother (1995) present the vocabulary of specialized technical and scientific areas as *extensions* of the core vocabulary of common language, with intermediate layers of 'common scientific' and 'common technical' vocabulary in between. At the extremes, they put for example business areas such as production management and corporate finance, next to mechanical engineering and computer science. New specialist registers are constantly added to the vocabulary of a language, so the extreme ends of their vocabularies will continue to drift apart.

These observations allow me to assume that if in a certain domain the people use a word that is not being used outside the domain, chances are high that this word represents a concept that is particularly important inside the domain and that has accumulated some special features over time. Given the importance of the domain vocabulary, we can assume that this same vocabulary will be pervasive in any information system that should be developed for the domain. Pervasive not only in the user interface (at the surface level of the system where NL words are used to refer to certain deeper concepts), but also buried deeply inside the system at the conceptual level.

[2]Syntax in the sense of frequency distributions of various constructions.

2.2.2 Terminology in the Lexicon

In most cases, domain terminology in conceptual modeling is stored inside the models as plain strings, with no linguistic or other information attached. Essentially, these strings are used as references, not as carriers of meaning, and there are no provisions to reduce redundancy. An example of such a system is InfoModeler (Asymetrix, 1994).

A straightforward improvement would be to eliminate all written strings in all models and to replace them by pointers to a central simple string repository. This is a well-known way of centralizing values, comparable to storing fixed program values in `#define` statements instead of scattering them all around the source file(s). However, such a simple repository still offers considerably less functionality than a Lexicon would do. We want not only to reduce terminology redundancy, but also to increase terminology coherence and its internal structure.

A 'true' Lexicon contains conceptual (meaning) information as well as morphosyntactic (form) information. The conceptual information can help to prevent that wrong words or word senses find their way into the conceptual model. An example of such a mistake is the overlooking of homonyms or the over-unification of similar words, where the same terminology is used for concepts that are in fact not compatible, although at first sight they might seem to be. Such a case can have two causes.

First, the domain specialists might have an incomplete understanding of their own domain and erroneously use the same terminology for distinct concepts. A typical case of such an error is the mixing up of type and instance level, e.g., by talking about *articles* and *an article* without recognizing that both expressions have different meanings. In this case, it is not only a matter of proper quantification, but also of intension and extension. This type of confusion is especially hard to track down because often the domain specialists are the only people who know the domain intimately. Professional analysts will usually have enough experience to catch many of these typical modeling mistakes, and many modeling paradigms are developed to make a clear distinction between type and instance level, exactly to prevent such mistakes. But an improper understanding of a domain will inevitably lead to this type of error.

The second type of error that causes over-unification is not a fault of the domain specialist, but of the model specialist. Some modeling paradigms use concept definitions that are less intuitive as might seem at first sight. While shared terminology suggests that the paradigms view the modeled concept in the same way, they might not. A salient example is the different world view of NIAM versus KISS. The sentence 'Customers can open accounts' in a commercial bank context can be interpreted in both paradigms. NIAM, being data-oriented, concentrates on the nouns. It would see this sentence as a fact type and deduce that there can be relations between customers and accounts, leading to questions about how many accounts one customer can open, how both customers and accounts are identified and so forth. KISS, an action-oriented paradigm, notes the action 'to open' and subsequently asks for more information about this action, such as its result (the opening of the account), which subjects (people) are allowed to perform it, when they are allowed to perform it (the dynamic work flow features), and whether the action is part of a fixed procedure.

In the end, both modeling approaches might have accumulated enough additional information to overlap, but especially in the early phases of a modeling project, misunderstandings about the *exact* interpretation of the terminology in both world views are bound to surface.

What stays constant in all modeling paradigms that take their terminology from the Lexicon is the combination of subsets of terminology; in the example, the combination of *customer*, *(to) open*, and *account*. Given the natural language used to paraphrase the paradigms, the particular domain the words are used in, and the 'common sense' meaning of the words, there is not much room for misunderstanding the meaning of the sentence when uttered in the domain itself.

On top of this, it should be possible to analyze the acquired terminology, to reorganize its structure without compromising the dependent applications, and to compare and merge several previously unrelated sets of domain terminology. A store of plain word forms (strings) alone is not sufficient to fulfill all these demands and wishes.

2.3 An Example

To illustrate the various types of information that a lexical analysis of a requirements document can provide, I will use a part of the ISO 'car example' (van Griethuysen et al., 1981, p.4-1).

> There are a number of manufacturers, each with one unique name. Manufacturers may start operation, with the permission of the registration authority (which permission cannot be withdrawn). No more than five manufacturers may be in operation at any time. A manufacturer may cease to operate provided he owns no cars, in which case permission to operate lapses.
>
> A car is of a particular model and is given a serial number by its manufacturer that is unique among the cars made by that manufacturer. The manufacturer is registered as the owner of the car as soon as practical. At this time it is given one registration number, unique for all cars and for all time. The year of production is also recorded. During the month of January only, a car may be declared to have been produced in the previous year. Eventually a car is destroyed and the date of destruction is registered. The history of a car must be kept until the end of the second calendar year following its destruction.
>
> [...]

Completely automatic lexical analysis without some form of pre-processing is limited in what can be accomplished, as will be shown in the following.

A simple lexical scan can quickly isolate the mainstay of the domain terminology used in this example, by listing all found nouns, verbs, and adjectives/adverbs. It reveals some concept forms (manufacturer, registration authority, car, serial number...) and frame forms (operate, given, registered, record, declare, destroy...).

Closer inspection also shows indirect concepts (referents) (he, that, it...) and derived words (destruction, operation).

One of the most straightforward automatic things then to do is to *normalize*, *stem*, or rather *lemmatize* the various words, e.g., by converting all nouns to singular and all verbs to infinitive. But for some words this poses a problem: 'name' can be both a singular noun and the infinitive or first person singular of the verb 'to name'.

The same basic mechanism of form look-up can be used to scan for derived words and combine them: 'destruction' is the nominalization of 'to destroy', and the Lexicon can provide this sort of links from either its concept base or through standard morphologic derivation rules. However, again there may be pitfalls here which suggest links between words that are semantically not there.

Unifying all equally formed words (after normalization) may look attractive, but poses a risk, especially for larger or multi-source texts. Homonyms and over-unification can introduce errors that are hard to pinpoint. It seems better to leave all referent solution, including the elimination of anaphora and personal pronouns, to syntactic analysis. Solving synonyms cannot be done by either lexical or syntactic analysis; for this, semantic analysis is required.

Clearly, simply using the plain conceptual Lexicon without additional help will not get us very far. This is where the *frames* in the Lexicon's concept base can help. When a given verb is related to its frame, e.g.,

```
give(ag human)(go thing)(rcp thing)
```

it is possible to scan for fitting nouns in the sentence. Experience gained in the Grammalizer Project (Hoppenbrouwers et al., 1997a) has shown that this is easier said than done, especially in sentences that were not written with this type of automatic 'parsing' in mind. But the predefined, fixed frame structures *suggest* word combinations that deserve to be inspected more closely, either by an automated NL parser or by a human analyst (by means of an appropriate graphical interface).

When most words have been gathered into frames, much more processing capabilities become available. Normalizing now is not hampered by choices between noun and verb, and this in turn reduces possible homonym and over-unification problems. Add to this the roles in the frames, e.g., the first frame slot of GIVE must be a HUMAN, and even more restrictions apply. The more concepts and frames have been acknowledged by the (human or machine) analyst, the more valuable the information in the Lexicon becomes.

The building of this semantic network of interrelated words is, of course, equivalent to conceptual modeling, and an experienced analyst will be better at it than a beginner. But instead of working with abstract, theoretical notions, she can work with familiar words and gradually build familiar sentences. When the particular conceptual modeling paradigm that eventually takes over is related to natural language, such as is the case with KISS, mapping of the semantic network acquired by lexical, syntactic, and semantic language analysis to the CM paradigm is relatively straightforward (Hoppenbrouwers et al., 1997a; Hoppenbrouwers et al., 1997b).

2.4 Other Uses of the Lexicon

Besides being of help during requirements elicitation (out of NL texts) and the subsequent conceptual modeling, a Lexicon can be used on existing conceptual models to verify them against 'common usage' of concepts and frames. Although true modeling errors are still difficult to find by using only lexical information, obvious irregularities can be tracked down, such as verb frames missing certain roles. For example, 'at this time it is given one registration number' might lead to the following frame:

```
give(ag --)(go car)(rcp registration_number)
```

where obviously the agent is missing and should be added. Type conflicts between the selection restrictions on frame roles and the actual types of role players in the conceptual model are another example of modeling errors that can be detected through application of a Lexicon (Hoppenbrouwers et al., 1997b).

The Lexicon could also be used to follow terminology links to other, previously constructed conceptual models, in order to quickly inspect them for possible reuse. If the frames overlap and the selection restrictions match as well, chances are that this particular part of the domain has been previously modeled, or that another, closely related domain has already been modeled and can be partially copied.

As a side effect of its original purpose (to decrease terminology redundancy and solve terminology clashes), the Lexicon will give its users a common framework to look at several potentially different paradigms, since much of the terminology of a domain modeled with multiple paradigms will be shared and thus can be used to unify concepts in otherwise unrelated paradigms. To use a database analogy, some *lexicon warehousing* and On-Line Analytical Processing (OLAP) might prove quite interesting during organization analysis, independent of the day-to-day system development tasks.

The same holds for inter-organizational lexicons, which can reveal overlaps and gaps between organizations that otherwise would be hard to discover in the early stages of analysis. This is especially true when the organizations in consideration have been using different paradigms for their information architecture descriptions. The (often technically influenced) views of their systems might differ, but they both have to speak the language of their trade.

Lastly, when we take a conceptual model, leave out all actual terminology (the orthography) and only retain the concept and frame structure, i.e., we drop the words but leave the structure in place, sophisticated pattern matching techniques might be able to discover overlap in structure in conceptual models that have no terminology in common at all. This might be the case when the models have been annotated in different languages.[3] Availability of a multi-lingual Lexicon that shares the same conceptual structures between lexical fields could increase the usability of the result.

[3]Note that frames do not imply a particular word order.

Chapter 3

The Structure of the Lexicon

— in which the author digs into the literature to discover what other researchers came up with. He performs an analysis of the Lexicon features that are considered essential by linguists, and compares them with views of cognitive psychologists, psycholinguists, and AI *researchers. —*

In linguistics and related fields, much work has been done on lexicography and other lexicon-related subjects. It would be rather pretentious to assume that I can give an even remotely comprehensive overview of this work. However, given the limited scope of this thesis—I concentrate on the use of lexicons in conceptual modeling—I can pinpoint several interesting areas which deserve further investigation. This chapter focuses on these aspects of lexicography and lexicons in general. I collect the essential characteristics of most approaches and select those features which I consider relevant to conceptual modeling. These selected features then will drive the design of the Lexicon Management System, as presented in Chapter 4.

3.1 The Lexicon in NLP

A Lexicon is not simply a collection of concepts words without any structure. Although some early linguists considered the Lexicon to be a mere list of basic irregularities, the place of exceptional cases, organized as an alphabetic list (Bloomfield, 1933), current insights in lexicography and computational linguistics leverage the role of the Lexicon in the whole NLP environment (Briscoe, 1991).

Boguarev and Briscoe (1989) discern the following basic types of knowledge that any Lexicon should contain to be relevant to NLP:

1. Phonological knowledge concerning the sound system and structure of words and utterances.

2. Morphological knowledge about the internal structure of words.

3. Syntactic knowledge of words concerning the organization of words into phrases and sentences.

4. Semantic knowledge concerning the meanings of words and how these meanings are combined to form the meaning of sentences.

5. Pragmatic or 'encyclopedic' knowledge which is central to many distinct tasks which an NLP system might undertake.

Except for phonological knowledge, all these knowledge types seem relevant for conceptual modeling as well. Whereas the first three knowledge 'levels' are quite distinct, Boguarev and Briscoe argue that there is no clear division between lexical semantic knowledge and more general pragmatic knowledge. An example of semantic knowledge they come up with is "that the concept denoted by *hit* involves two entities, an agent (the hitter) and patient (the person or thing being hit), and that these entities will be supplied by the subject and object of the verb, respectively." Pragmatic knowledge according to them is used for "recovering the referents of pronouns, reconstructing elliptical utterances or analyzing the speaker's presuppositions or communicative intentions which lie behind a particular utterance."

Storage of pragmatic knowledge, sometimes also called 'common sense knowledge,' still poses many problems (Lenat, 1995a), and in this thesis I will mostly discuss the morphological, syntactic, and semantic levels of the Lexicon.

3.1.1 The vocabulary

In order for any NLP engine to process a random text reliably, its vocabulary (the extension of the set of lemmas in the Lexicon, i.e., the *fund* reduced to root forms) should approach that of a human being. Common estimates for the number of lemmas available to humans range from a few thousands to several tens of thousands. Vocabulary is not static; especially in the technical-scientific register, the available vocabulary grows with about 6000 lemmas per language per year (Ulijn and Strother, 1995, p.101). Guiraud (1959) observes that the 1000 most frequent words suffice for 86% of a text and that the 4000 most frequent words are enough to account for 97.5%, but even a few missing key words may sent a NLP system up the garden path. A practical goal for NLP systems seems to be about 10,000 lemmas, but: "The common-wisdom figure of 10,000 roots is misleading. Word count (e.g., how are multiple senses counted?) and degradation tolerance vary from system to system. The question boils down to the value added per each added word." (Zernik, 1991, p.22).

The NLP systems that actually got built often did not even remotely reach this goal. In 1987, a workshop on linguistic theory and computer applications reported on an informal poll to establish the average size of the lexicon used by the prototypes discussed. Taking into account a 5000–6000 word vocabulary used by a machine translation system, the average lexicon size came to 1500 words. Without this vocabulary, the average size was about 25 (Boguraev and Briscoe, 1989). Compare this with the Oxford English Dictionary, available on-line, which contains roughly 250,000 independent words.

On-line resources such as the Oxford English Dictionary (OED) or the Longman's Dictionary of Contemporary English (LDOCE) seem a good solution to acquire many lemmas relatively easy, but methods to extract a Lexicon out of machine-readable dictionaries[1] (MRD) pose the difficulty that the MRD itself is an NL text—a chicken-and-egg problem. Various projects have been trying to capture the phonological,

[1] A dictionary is 'a printed word book for human readers' (Guthrie et al., 1996).

syntactic, and semantic information that is, either implicitly or explicitly, available in MRDs (Copestake et al., 1992; Vossen, 1995). A comprehensive overview is presented in Guthrie et al. (1996).

Despite all efforts, no Lexicon will ever contain *all* possible lemmas in a language, because a language and its users are highly dynamic. Not only can many lemmas be modified by means of prepositions or particles (the set of all basic and derived lemmas combined is called the *fund* of a language), but the list of different senses of words goes on and on: "... any finite enumeration of word senses will not account for creative applications [...] in the language" (Pustejovsky, 1993).

There is little hope to include all truly new lemmas that are coined, but some new lemmas are actually composed out of existing ones by deterministic predicate formation rules. These rules cause the fund of a language to be infinite. Weigand and Hoppenbrouwers (1997) discuss various mechanisms which could extend a static Lexicon to dynamically include the derived lemmas.

In the context of conceptual modeling, certain predicate formation rules such as verbal nominalization (Weigand, 1990, p.88) and, in a sense, compounding (Section 3.3.3) could be used to suggest links between words that are found in a conceptual model. Eventually the derived words must be entered into the Lexicon explicitly to confirm the proper frames etc., and also to give the information system's Data Dictionary or Repository a solid handle on the word, but elicitation of these frames might be facilitated by controlled application of predicate formation rules during knowledge acquisition.

3.1.2 The taxonomy

Through links between words, either in a form of super/subtype (taxonomy) or in synonymy, antonymy, and other relationships (thesaurus), many aspects of word usage and meaning can be captured without the need for formal or informal definitions.

Researchers such as Sparck Jones and Amsler and White did a lot of work on Lexicon organization, especially by clustering groups of words with common features. However, they either used a pre-existing taxonomy to cluster the words (Amsler and White, 1979), or followed statistical procedures which suffered from the regular problem of all automatic classification techniques: that the clusters, when found, did not have meaningful names (Sparck Jones, 1964).

The currently most common way of looking at dictionary taxonomies is inspired by the fact that for each head word, most dictionaries present a *genus term* and one or more *differentiae*. An example of this would be: "**grill room**, a restaurant (genus) that makes a specialty of grilled foods (differentia)". This leads to a structure where each head word 'is a' head word plus some differences—a tree. But even the work of Amsler already produced a structure best described as a 'tangled hierarchy,' because not all words can be fully defined as a genus/differentiae pair (Guthrie et al., 1996). The current semantic networks also use this tangled principle, and often add other structures. The LDOCE, for example, presents both a taxonomy based on analytical definitions ('abstract,' 'concrete,' 'animate,' ...) and a classification by subject ('engineering,' 'geology,' ...) (Procter, 1987).

More recently, taxonomies based on statistics and mutual dependencies have

surfaced (Vossen, 1995), which are less theoretically and more empirically inspired. See Section 3.4.2 for a more extensive discussion of these approaches, including some taxonomies that are driven by non-linguistic ideas.

3.1.3 Lexicon Representations

Many lexical theories of grammar use a particular theoretical framework called a *feature structure* (Minsky, 1975; Aït-Kaci, 1986; Pollard and Sag, 1987) to convey lexical and grammatical information. A convenient notation for feature structures is a typed attribute-value matrix with nested sub-feature structures and co-indexing. Usually these structures are drawn with nested square brackets, but a symbolic linear notation is also possible. In this thesis, I will use a simplified form of the former, as follows:

$$\begin{bmatrix} \text{attr1} = \text{val1} \\ \text{attr2} = \begin{bmatrix} \text{attr2a} = \text{val2a} \\ \text{attr2b} = \text{val2b} \end{bmatrix} \\ \text{attr3} = \begin{bmatrix} \text{attr3a} = \begin{bmatrix} \text{attr3a1} = \text{val3a1} \\ \text{attr3a2} = \text{val3a2} \\ \text{attr3a3} = \text{val3a3} \end{bmatrix} \\ \text{attr3b} = \text{val3b} \end{bmatrix} \end{bmatrix}$$

Briscoe (1991) argues that one advantage of having a typed lexical representation language in a large collaborative project is that once an agreed type system is adopted, the compatibility of the data collected by each site is guaranteed, whereas in an untyped feature system, typographical errors and so on may go undetected.

Authors such as Carpenter (1990) proposed typed feature structure schemes in which there also is a *partial ordering* on types. Schemes such as these can be used to define an inheritance hierarchy in which feature structures 'lower' in the hierarchy are monotonically enriched with information derived from 'higher' levels. More work on typed feature structures and inheritance hierarchies can be found in (Calzolari, 1993a) and (Daelemans and Gazdar, 1990).

Another major framework for Lexical representations is DATR, a simple, Spartan language for defining nonmonotonic inheritance networks with path/value equations, one that has been designed specifically for lexical knowledge representation (Evans and Gazdar, 1996). DATR was developed to provide a natural way of saying "this lexeme is regular except for this property," which is a typical example of a problem that can elegantly be solved by inheritance networks (Daelemans et al., 1992). The language follows a minimalist approach, and thus lacks many of the constructs embodied either in general purpose knowledge representation languages or in contemporary grammar formalisms. Yet it is sufficiently expressive to represent concisely the structure of lexical information at a variety of levels of language description (Evans and Gazdar, 1996).

DATR uses *paths* of *atoms* to identify *values*, as in

```
<syn cat>  == verb
```

```
<syn type> == main
<syn form> == present participle
<mor form> == love ing.
```

Here, angle brackets < ... > delimit paths, and values can be atomic or they can consist of atom sequences. Values can also refer to existing DATR definitions, indicated by double quotes "...". A DATR representation of the partial analysis of English verbs might look like:

```
VERB:
  <syn cat> == verb
  <syn type> == main
  <mor form> == "<mor "<syn form>">"
  <mor past> == "<mor root>" ed
  <mor passive> == "<mor past>"
  <mor present> == "<mor root>"
  <mor present participle> == "<mor root>" ing
  <mor present tense sing three> == "<mor root>" s.

Love:
  <> == VERB
  <mor root> == love.

Word1:
  <> == Love
  <syn form> == present participle.

Word2:
  <> == Love
  <syn form> == passive participle.
```

DATR provides for many powerful inheritance and reasoning mechanisms and is easily implemented, usually in Prolog. Also, DATR is a *language* for lexical knowledge representation, not a theoretical framework for the Lexicon (Evans and Gazdar, 1996).

My own solution for a practical way to build Lexicons through a generic storage engine, a Lexicon Management System (see Sections 3.3–3.4 and Chapter 4), shares many of the views of DATR, such as independence of any theoretical frame work (although Functional Grammar influenced at least the examples) and focus on inheritance to reduce redundancy. However, I also tried to incorporate methods based on database technology to improve the feasibility of the LMS for use in large-scale projects. Implementations of languages such as DATR often fall short in important areas such as multi-user management and code/data separation, especially when written in Prolog where code and data can easily be unified. Efficient search mechanisms for lexical forms, retrieval of closed sets, and server-based access control are some other reasons why a pure DATR implementation might not be sufficient for a large-scale LMS.

3.2 The Lexicon in FG

Because Functional Grammar (FG) plays a central role in the theoretical base of the Lexicon (Chapter 4), it seems appropriate to discuss the lexicon part of FG in more detail. Weigand considers a conceptual model, such as a NIAM model, to be not much different from what is traditionally stored in a (FG) Lexicon. He lists some reasons why a Lexicon offers extra functionality beyond the common conceptual modeling paradigms (Weigand, 1990, p.75). The remainder of this paragraph is heavily inspired by his work.

The Lexicon Management System is based on both Dik's original Functional Grammar (1989), and its more specialized descendant CPL by Dignum (1989). They both start with *terms*, *predicates*, and *predicate frames*, which can all be stored in a Lexicon and are the basic building blocks for all expressions. For the sake of this short explanation, it suffices to equate terms with nouns, predicates with the roots of verbs, and predicate frames with the various senses of the same verb (such as with *to bar*; see Section 3.2.1).

When we take a predicate frame, fill in its slots with appropriate terms, and give it a position in the temporal-spatial continuum, we create a *predication*. A predication as a whole designates a *State of Affairs* (SOA), something which can be the case in some world. By applying a modal operator to a predication we form a *proposition*, a possible fact. Propositions can be true or false in a certain world. They can be viewed as statements about the world, and can be exchanged between (linguistic) agents. When we combine a sequence of propositions exchanged between agents into a (partial) discourse, using the basic operators of speech act theory, we reach the *message* level, the highest level of Functional Grammar. Given the aim of this thesis, I will not discuss the expression rules that FG uses to convert a formal, semantic FG formula into an utterance in a real, natural language. These expression rules are language-dependent and the subject of ongoing research. For a discussion of these expression rules in conjunction with the Lexicon, see Weigand and Hoppenbrouwers (1997).

3.2.1 Basic FG Lexicon Elements

Following the FG theory, the (FG) Lexicon contains three sets (Weigand, 1990, p.78):

1. *predicates*: grammatical information
2. *predicate frames*: semantic information (roles and selection restrictions)
3. *predicate schemata*: conceptual information (pragmatic example cases)

The *predicates* represent the words, independent of their usage. For example, the word 'bar' in English equals the FG predicate BAR. Weigand specifically excludes pronunciation and spelling, word category (e.g. 'transitive verb'), and irregular forms from the grammatical information. This reinforces his view of the Lexicon as mainly an addition to conceptual models, but clearly, the less conceptual lexical and morphosyntactic information should be stored as well.

The same predicate can usually be associated with several *predicate frames* (Weigand, 1990, p.78), in the case of BAR:

```
bar(ag human)(go door)
bar_in(ag human)(go animate)
bar(ag human)(go way)
bar(ag human)(go human)(so soa)
bar(ag human)(go soa)
bar(go thing)
```

Each of these predicate frames gives both the semantic roles and the selection restrictions on these roles, e.g. the first example conveys that there is a particular way of using BAR in which some object of type *human* initiates the action 'to bar' on some object of type *door*. As Weigand shows, the predicate of a frame can be replaced by another predicate (while retaining the frame) if the need should arise. In a sense, such equivalent predicate frames indicate synonymity between predicates when used in a given frame. For example, BAR in the third sense as listed above can be replaced by OBSTRUCT.

Prototypical usages of the frame which give meaning definitions, such as 'a sky barred with clouds' (`bar(go sky)(instr cloud)`), are called *predicate schemata* or (following Carnap) *meaning postulates* (Vossen, 1995, p.19). Note that there may be many predicate schemata subsumed under one predicate frame. Only the top element of this structure is called a (lexical) frame, the other ones are called schemata (Weigand, 1990, p.79). Predicate schemata in the Lexicon can be regarded as examples of real-world predicate frame usage—just as the (example) population of a database table in conceptual modeling. Alternative approaches to meaning postulates can be found in explicit relation modeling (Sowa, 1983) and in rich feature systems (Pollard and Sag, 1987).

3.2.2 Theory-driven Taxonomy

According to Weigand, FG theory suggests a particular taxonomy of the Lexicon. Apart from using various more or less arbitrary Lexicon organizations, such as by Aarts' primary features (Aarts, 1976) or by the semantic codes of Longman's LDOCE (for more artificial category systems, see Section 3.4.1), FG theory itself can be used to *deduce* a taxonomy (Weigand, 1990, p.99). Starting from FG notions such as first-order entity, second-order entity (State of Affairs), Action, Process, Position, and State, a taxonomy emerges which can be used to define the top levels of a Lexicon (see Figure 3.1). Weigand prefers to give these *primitive types* suggestive names ('thing') rather than abstract ones ('first-order entity'), but the types stem from theory, not from observation:

> "(...) We assume that the upper layers of the taxonomy are theoretically defined. The elements occurring there are called *primitive types* and we do not assume that their names necessarily correspond with their usage in natural language (...). Their definition is given by the linguistic theory."
>
> (Weigand, 1990)

Some examples of each primitive type are:

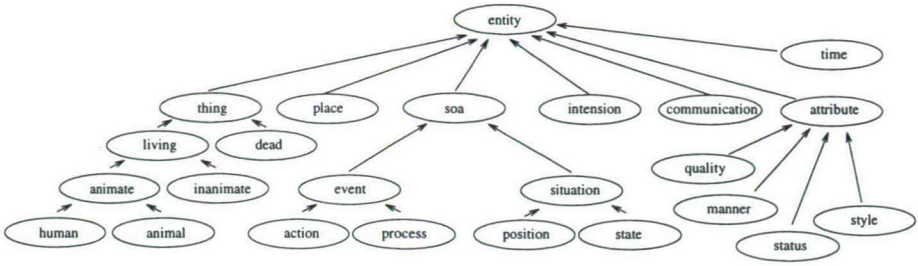

Figure 3.1: A Theory-driven Taxonomy of the FG Lexicon

human: employee, man, woman, dwarf, child, company.
animate: fish, dog, dodo, pet.
living: animal, tree, E.Coli.
material: rock, fuel, star, house, vehicle.
thing: circle.
action: production (produce), transaction (give, sell), comparison, game.
process: rain, corruption (corrupt), fall.
state: sleep, anger, existence.
position: management (manage).
intension: desire, idea, belief, theory, concept.
communication: word, symbol, sentence, language.
place: in(), between().
time: day, hour, winter.
quality: temperature, number, color, size.
manner: speed, democracy, intensity.
status: probability, truth.

3.2.3 Morphosyntax

Whereas in FG and many other formalisms the identifier of the predicate (such as BAR) implies the root form of the word, which is notationally convenient, practical applications require a more flexible storage mechanism for morphosyntax. I propose to separate the traditionally combined morphosyntactic and semantic features into a *lexical base* and a *concept base*, which are connected by bidirectional links. Sections 3.3 and 3.4 expand on these topics; here I will discuss the relation between both of them.

The concept of *signe*

Already in the early years of linguistics, De Saussure (1916) made a distinction between *signe*, *signifiant*, and *signifié*. I will cite him[2] because his explanation is right to the point.

[2]Actually, the people who put his *Course* on paper.

> "Le signe linguistique est donc une entité psychique à deux faces [...]. Ces deux éléments sont intimement unis et s'appellent l'un l'autre. Que nous cherchions le sens du mot latin *arbor* ou le mot par lequel le latin désigne le concept 'arbre,' il est clair que seuls les rapprochements consacrés par la langue nous apparaissent conformes à la réalité, et nous écartons n'importe quel autre qu'on pourrait imaginer.
>
> Cette définition pose une importante question de terminologie. Nous appelons *signe* la combinaison du concept et de l'image acoustique: mais dans l'usage courant ce terme désigne généralement l'image acoustique seule, par example un mot (*arbor*, etc.). On oublie que si *arbor* est appelé signe, ce n'est qu'en tant qu'il porte le concept 'arbre,' de telle sorte que l'idée de la partie sensorielle implique celle du total.
>
> L'ambiguïté disparaîtrait si l'on désignait les trois notions ici en présence par des noms qui s'appellent les uns les autres tout en s'opposant. Nous proposons de conserver le mot *signe* pour désigner le total, et de remplacer *concept* et *image acoustique* respectivement par *signifié* et *signifiant*; ces derniers termes ont l'avantage de marquer l'opposition que les sépare soit entre eux, soit du total dont ils font partie."

In short, De Saussure proposes to give the name *signe* to the combination of a mental image (*signifié*) and the corresponding language expression (*signifiant*). He only considers 'acoustic images' to be *signifiants*, which is in line with the general idea that language, by nature, is an auditive system. In the information system world, however, written (textual) expressions are much more commonplace, and therefore I choose to extend the meaning of *signifiant* to include written words. Additionally I want to attach more *signifiants* to the same *signifié*, to grasp the concept of synonymy and to enable multi-domain and multi-language support.

Lexicals and Concepts

I propose to view the mental image as the core and to assign possibly several physical images to the mental image. As such, I do not use the *signe* 'catch-all' term, but I continue to use the traditional *concept* term for the mental image and state that a *concept* can have various associated *lexicals* to express it in (written) language. In other words, I do not combine the concept and its lexicals into one whole.

There has been much other work on the same subject. For example, Aristotle says:

> "Spoken words are symbols of experiences ($\pi\alpha\theta\acute{\eta}\mu\alpha\tau\alpha$) in the *psyche*; written words are symbols of the spoken. As writing, so is speech not the same for all peoples. But the experiences themselves, of which these words are primarily signs, are the same for everyone, and so are the objects of which those experiences are likenesses."
>
> (*On Interpretation*, 16a4).

Ogden and Richards (1923) presented the same basic idea in what they call the *meaning triangle* (see Figure 3.2). The left corner is the *symbol* or *word*; the peak is

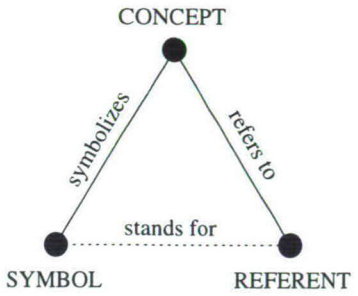

Figure 3.2: The Meaning Triangle

the *concept, intension, thought, idea,* or *sense*; and the right corner is the *referent, object,* or *extension.* For some concepts, one corner of the triangle may be absent: a person may have a concept of an object for which he knows no word, or he may have a word for a concept that has no extension. The word 'unicorn' is mapped to the concept UNICORN in the same way that 'horse' is mapped to HORSE, even though there are millions of horses in the world, but no unicorns (Sowa, 1983, p.11). In FG, Dik states that entities (concepts) are not 'things in reality,' but 'things in the mind' (Dik, 1989, p.113) and gives three reasons why entities must be assigned mental status (the equivalent of concepts):

1. There are many things which we can refer to and talk about, but which do not exist in reality. For example, consider a sentence such as 'Last night I dreamed of ants as big as dogs.'

2. We can refer to 'real' things only to the extent that we have some mental representation of them. Referring somebody to the Eiffel Tower will only be successful if (s)he has already a mental 'picture' of the intended referent.

3. We can refer to and efficiently talk about things in reality even in situations in which these things are nowhere to be perceived or otherwise directly experienced.

All these reasons are linguistically and philosophically motivated. I want to add a few reasons why it is useful to also separate the symbols (lexicals) from the concepts and referents:

1. It facilitates translation of domains to other domains or languages. Although not all concepts in one domain/language have a direct 1-to-1 correspondence with concepts in other domains/languages, many have, especially when they refer to physical entities.

2. In information technology, it is only possible to work with (physical) *representations* of real-world objects. Systems store and manipulate data *about* concepts, not the concepts themselves. Therefore, IT systems store symbols (more precisely, the equivalent of proper nouns).

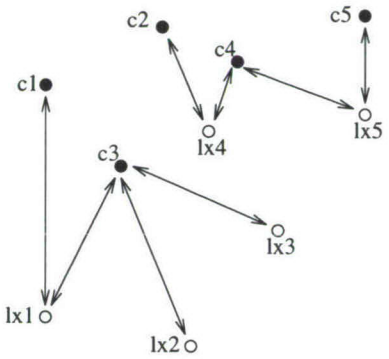

Figure 3.3: Linking Lexicals and Concepts

3. In information analysis, the distinction between types and type instances is paramount. When I talk about the concept PERSON, it must be made absolutely clear that I am not at all interested in the orthography 'person,' but in the mental concept instead.[3]

4. The orthography (symbol) of a concept will often follow language-dependent morphological rules, such as plural formation. It is impractical to use the identifier of the concept such as TREE both for identification and morphology generation.

Lexicals and Concepts in the Lexicon

A simplified illustration of the lexical/concept system is presented in Figure 3.3. By means of *linkers*,[4] this partial Lexicon contains the information that a certain concept C3 (which can be either a term or a frame) can be expressed by three lexicals LX1, LX2, LX3; in other words, the three lemmas LX1, LX2, LX3 are synonyms. LX4 services the two concepts C2, C4, and therefore introduces a homonym. The lexicals carry *no meaning at all*, they only represent a single lemma orthography (a root) and can be grouped in classes that share the same regular morphological expression rules.

This scattered approach to lexical information storage offers many advantages above a combined morphosyntactic/semantic system. It is no longer necessary to store multiple word entries repeatedly; as long as these words follow the same morphological rules and have the same exceptional forms, they can be shared. Implementation of the specific requirements for large-scale lexicons becomes simpler and more efficient. The flexible structure allows for interesting combinations between concepts, e.g., crossing domain and language boundaries where appropriate.

For example, Weinrich (1964) made a perceptive distinction between two forms of ambiguity: *contrastive* and *complementary*. This distinction classifies homonymous

[3]There is another distinction between type and type instance, but that plays at the analysis level, not at the concept/word level.

[4]In the TREVI project now called *denotations*.

nouns such as *bank* as having two contrastive senses of a lending institution and a river's edge. The same noun 'bank' has complementary senses of the institution and the building, as well as the people in the organization (Pustejovsky, 1991). In many lexical storage systems, the only way to store multiple word senses is by sense duplication (or by a meaning postulate). Such a Lexicon suggests a *monomorphic language* (Strachey, 1967), in which lexical items have only one type and denotation. Lexical ambiguity then is handled simply by a multiple listing of the word in the Lexicon, with no link in between. The scheme as presented in Figure 3.3 allows a much clearer setup, with the actual English noun 'bank' (including all morphological forms) being the lexical and the various senses the concepts. That some concepts are complementary ambiguous can then be expressed by adding more links (through frames) between the concepts. These linking frames each can have one or more lexicals, the equivalents of English verbs.

Another indication of the proposed scheme's flexibility is the following. The lexicals carry all morphosyntactic information, including word type such as *English noun* or *French verb* (this is necessary to generate the regular word forms). The concepts only carry meaning, such as a place in a taxonomy or as a role frame with selection restrictions. But nothing prevents us from attaching a noun lexical to a frame concept, or a verb lexical to a noun concept. In this way, a lexicographer can add various interesting features, e.g., if a frame concept has a noun lexical available (next to its normal verb lexical), this noun lexical can be used for nominalization instead of a standard rule that tries to generate a noun out of the verb's root.

For a more thorough discussion of this linking and framing mechanism, see Chapter 4.

3.3 The Structure of the Lexical Base

The lexical part of the Lexicon contains word form information only. A straightforward, but memory-consuming way to implement a form base would be to store all forms for every available lexical, e.g., 'book, books'. But due to the regular aspects of morphology, many word forms can be deduced from a single root form, e.g., English plural is ROOT+'s'. Using this approach, all regular[5] word forms can be deleted from the lexical base and replaced by inference rules in the management system, while only irregular forms (and the root form) have to be stored in full.

A rules-with-overriding-plain-forms approach is very well suited for an object-oriented architecture. In the next section some more advantages of OO in building a Lexicon are discussed.

3.3.1 Inheritance and Object-Orientation

Daelemans argues that the use of object-oriented techniques, including inheritance, can be as profitable to linguistics as it is to software engineering (Daelemans, 1990).

[5]Not necessarily regular in the linguistic sense; every set of 'irregular' forms that can be covered by a rule can be included. It is a matter of choice when to implement a rule and when to store plain forms.

In his paper, he mostly concentrates on the usage of OO techniques for building linguistic formalisms, but I will quote the relevant parts about what OO and inheritance can mean for the representation of linguistic knowledge:

> "When using an OO approach to the modeling of linguistic knowledge, inheritance hierarchies fulfill several related functions.
>
> **Generalization.** Representation of markedness differences, the *regular - subregular - irregular - exceptional* continuum, and blocking effects.
>
> **Information Sharing.** Minimization of redundancy through information sharing.
>
> **Default reasoning.** Reasoning with incomplete knowledge.
>
> **Knowledge Integration.** Combining and integrating knowledge from different sources (multiple inheritance).
>
> The methodological attraction of using an OO formalism is that both linguistic and non-linguistic knowledge, and all types of linguistic knowledge (from acoustic to pragmatic) can be represented in a uniform way. Artificial boundaries between e.g. world knowledge and semantic knowledge, morphological and syntactic knowledge, or even between object and meta knowledge, disappear this way. At the same time, modularity can be guaranteed by restricting the combination between objects and by using object types of different grain size. An OO model is modular at a local level, and interactive at a global level (where grain size determines what is local). E.g., the Lexicon as a whole can also be defined as an object type that can be specialized, and accessed through a clean interface."

The whereabouts of multiple inheritance, a notoriously difficult problem which can be approached in many different ways, are not yet satisfactorily solved in any approach I know. Many people agree that multiple inheritance (as opposed to simple or single inheritance) is a useful tool to enhance expressiveness and reduce redundancy: "The main advantage that multiple inheritance offers over single inheritance is the ability to inherit several (either orthogonal or complementary) sets of properties of classes in more than one path through the hierarchy." (Russell et al., 1990). But the proposed solutions to the ambiguous inheritance that pops up when more than one superclass offers a certain property while the current class does not, either ignore this problem (leaving the inheritance over to an implementation-dependent selection process, e.g. 'first class found by the tree climbing algorithm') or leave it completely to the lexicographer, who must decide on e.g. the superclass precedence order (Russell et al., 1990). The latter approach works reasonably well when building a Lexicon top-down, but when in already existing superclasses a property is added, this may lead to newly created multiple inheritance conflicts. An LMS should in such a case reject the update until the conflicts are properly resolved—which is a traditional database update problem, technically solvable, but still not automatic.

3.3.2 Example Lexical Class

Each lexical in the Lexicon is an instance of one of the lexical classes. Lexical classes, such as 'English Noun,' can be defined by a lexicographer in a dedicated language, called LIX (see Section 4.4). An example of the 'English Noun' lexical class would be:

```
lexical englishNoun (forms=singular,plural)
script englishNoun singular
  SCRIPT
  return root
  SCRIPT
script englishNoun plural
  SCRIPT
  return root+'s'
  SCRIPT
```

The Lexicon now can be filled with English noun lexicals as follows:

```
insert englishNoun (forms=(root='book'))
insert englishNoun (forms=(root='safe',plural='safes'))
insert englishNoun (forms=(root='police',plural=none))
```

Naturally, the plural formation script can be greatly enhanced to catch the cases such as with 'knife', where a slight adaptation of the standard rule ROOT+'s' is in order. Simple string manipulation can be used to implement a scan for the ending 'y' and produce 'hobb*ies*' instead of '*hobbys', among other things. The next example, based on Aarts and Aarts (1986, pp.24–25), shows a reasonably complete script for the plural formation of English nouns, written in a suitable language. Cases not covered by this script are considered exceptions and need to be stipulated in their Lexical instance.

```
script englishNoun plural
  SCRIPT
  plural := root+'s'
  str := last(root,1)
  if (str='s') or (str='z') or (str='x') then
    plural := root+'es'
  if (str='y') then
    plural := nolast(root,1)+'ies'
  str := last(root,2)
  if (str='sh') or (str='ch') then
    plural := root+'es'
  return plural
  SCRIPT
```

Powerful pattern-matching languages such as Perl might produce shorter scripts, but could also decrease readability. The actual scripting language should be independent

of the Lexicon implementation, so that each script can be written in the most appropriate language. This is the reason why the scripts are guarded by the SCRIPT sentinel keywords: they switch off the LIX parser so that other languages can be included in-line, without escaping sensitive characters such a quotes.

Further subclassing of the englishNoun lexical class can be used to produce specialized classes, similar to the approach as with with complex verbs (see Section 3.3.3).

3.3.3 Compounds and Complex Verbs

In languages such as German and Dutch, nouns and their modifiers can be stringed together to form a new, more specialized noun, e.g., 'vracht' and 'wagen' produce 'vrachtwagen' ('freight car'). Without special care, a Lexicon containing only the basic lexicals will not recognize these compounds. Several algorithms exist to use existing single lexicals to scan for compounds (Vosse, 1994), but they can only indicate a *possible* compound. To guarantee that the Lexicon correctly and authoritatively recognizes compounds, they must be confirmed by the lexicographer and thus be entered manually. In case of a detected possible (unconfirmed) compound, a good strategy would be to return the lexical that is last in the chain (Right Hand Head Rule), assume that this is the *genus*, and raise a flag to indicate unconfirmed compound resolution. Linguistic client software then could assume that the actual word is some kind of specialization of the concept associated with the genus lexical. This approach is taken in the Lexicon of the *Grammalizer* (see Appendix B). Since compounds can be formed at will by prefixing any number of adjectives, there will never be a situation where *all* compounds can be found directly, so something like this genus approach should be implemented in any case.

To (manually) confirm compounds, two approaches are possible. The first is to simply store the complete new orthography ('vrachtwagen'), with no special attributes whatsoever. This would work, but is not really appealing because the already existing lexicals 'vracht' and 'wagen' are not re-used in any way. A better approach would be to subclass the original 'noun' class and add room for a number of adjective lexicals in this 'compound (noun)' class. The compound noun 'vrachtwagen' then can be added by creating a new lexical of the class 'compound noun,' letting this instance inherit the genus lexical from the existing noun lexical 'wagen', and adding a pointer to the existing adjective lexical 'vracht'.[6]

Both approaches also require the creation of a specialized VRACHTWAGEN concept in the concept base (Section 3.4), although it could be argued that in the second case the explicit links between the genus and the differentia (the adjective) at the lexical level could be traced back to the modifiers at the concept level.

Multi-words (lexical units containing one or more spaces) such as 'zebra crossing' or 'bird of prey' can be considered partly compounds, partly idioms (Section 3.3.4).

A related case is the complex verb in English, e.g., 'fill in'. The verb part of this complex verb, 'fill', is regular, but the particle 'in' should be contained in the lexical as well. The preferred way of implementing the englishComplexVerb class in the Lexicon is by subclassing the already existing englishVerb class:

[6]The current Lexical class mechanism does not yet allow for indirect roots, i.e., using a pointer to another root instead of a plain string.

```
lexical englishComplexVerb (parent=englishVerb
                            forms=(particle))
script englishComplexVerb s1p
  # Singular, 1st person, present
  SCRIPT
  return root+' '+particle
  SCRIPT
script englishComplexVerb s2p
  # Singular, 2nd person, present
  SCRIPT
  return root+'s '+particle
  SCRIPT
    .
    .
    .
```

Note that the complex verb class has no ROOT form by itself; the ROOT form is inherited from the englishVerb class. As with the compound example, when there already is a verb 'fill', the complex verb 'fill in' can be created by making the 'fill' lexical a parent of the 'fill in' lexical so that the ROOT value 'fill' does not have to be repeated. Scripts are inherited between lexical *classes* while forms are inherited between lexical *instances*.

3.3.4 Idioms

Idioms (accepted phrases or expressions, having a meaning different from the literal, such as 'kick the bucket') could be handled in a way comparable to compounds, with the obvious difference that idioms not available in the Lexicon will inevitably send the parser up the garden path—there is no hope in inferring the actual meaning of idioms. Since it is possible to use standard morphosyntactic rules on idioms just as on 'normal' expressions ('they have kick*ed* the bucket'), storing the complete phrase as a multi-word is not attractive. The *combination* of certain lexicals is the key to successful idiom resolution, and this whole combination should be attached to a meaning postulate in the concept base. A dedicated 'idiom' Lexical class, which has options to include a complete expression separated in nouns, verbs, and adjectives, seems a possible solution for the idiom problem. However, since the true problem with idioms is not in their irregular lexical behavior but in their irregular *meaning*, idiom solving might best be left to the parser. The Lexicon in such a case provides a base of known idioms, but is not capable of detecting idiomatic language by itself.

In the context of conceptual modeling, idioms (together with metaphors and other non-literal language) are inconvenient and should be avoided where possible. A Lexicon that is used as an assistant in conceptual modeling therefore does not need to have full idiom capacities, although it will never hurt to equip it with a few phrases that are often encountered and should be flagged as being suspect.

3.4 The Structure of the Concept Base

Based on a definition by Dana Scott, Weigand (1990, p.77) makes a distinction between 'vocabulary,' 'thesaurus,' and 'dictionary:'

The vocabulary provides the official list of correct forms of words, presents only syntactical features, and gives idiomatic patterns of usage if necessary.

A thesaurus provides the official survey of correct terminology for concepts, presents only basic semantic features, structures the terms in a semantic net, and adds to the vocabulary special patterns of usage appropriate to the special concepts.

A dictionary presents the definitions of terms from the thesaurus, gives humans the understanding of specialized words, helps shape the growth of the thesaurus, and helps authorities in deciding on the admission of new terms.

The vocabulary is the morphosyntactic part of a Lexicon, as described in Section 3.3.

An important role of the Lexicon is to provide *organization* of terminology, to provide meaningful structures that are of use to information analysts and other users. To accomplish this, the thesaurus will need to be included, but the dictionary needs not. This implies that very few, if any, concepts in the Lexicon will be defined stand-alone in formal terms: most concepts are defined by their connections with other concepts. This view is in line with current insights in linguistics (Kyle and Woll, 1985), psycholinguistics (Ulijn and Strother, 1995), and cognitive psychology (Glass and Holyoak, 1986). Additionally, research in the fields of philosophy (Kaminsky, 1969) and artificial intelligence (Gruber, 1993) suggests that one single formal conceptualization of 'the world' is not to be expected for quite a while, so there will inevitably be overlaps and gaps in the stored knowledge.

Of course, this approach can lead to some confusion when somebody tries to acquire or capture the terminology of a new domain, but explicit language teaching, providing students with words and grammar to assimilate, has always been 'unnatural.' As Kyle and Woll (1985, p.174) say:

> "It is virtually impossible to *teach* a language; what one can hope to do is to provide a situation where the complex variables which contribute are maximized."

One of these complex variables clearly is the amount of background knowledge that the student has available. The more background knowledge, the higher the possibility that new concepts can be attached to already existing knowledge (Ulijn and Strother, 1995, p.129). Traditional (monolingual) dictionaries assume a high level of background knowledge because they present definitions in unrestricted natural language. Some language learner's dictionaries, such as Longman's LDOCE, explicitly limit the vocabulary used in the definitions: the Longman Defining Vocabulary contains 2000 words. This Defining Vocabulary was based originally on *A General Service List of English Words* by Michael West, the only frequency list to take into

account the frequency of *meanings* rather than the frequency of word forms (Procter, 1987).

What sets apart my Lexicon from word lists with NL definitions is my way of interconnecting words *mainly by their patterns of usage in a domain.* Analytical meshes of words such as WordNet (Miller et al., 1993) also interconnect words in a semantic net, but they aim at a much broader coverage of knowledge. My Lexicon is meant to support information analysis in IT projects, and limits itself to the vocabulary that is essential for the domain in question. This does not exclude a reliance on pre-wired semantic nets such as WordNet for re-use, yet enables an analyst to start entering words into the Lexicon without any pre-existing words.

3.4.1 Concept Ontologies

The Oxford English Dictionary defines *ontology* as 'the science or study of being.' In philosophy and metaphysics, the word *ontology* refers to a systematic account of Existence. As such, an ontology in philosophy is a description of the minimal set of concepts that a language needs to express all its other concepts. Various proposed ontologies exist to date, two of which are described in (Kaminsky, 1969), but in this thesis I will not focus on the philosophical issues of ontology.

There is also a usage of *ontology* in the Artificial Intelligence research community, where it usually means 'a specification of a conceptualization.' A conceptualization is an abstract, simplified view of the world. Therefore, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for a community of people or automated agents (Gruber, 1993). The term *Universe of Discourse* is often encountered to describe the set of all objects that can be represented. In an ontology, definitions associate the names of entities in the Universe of Discourse with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms.

In an AI context, the purpose of an ontology is to enable knowledge sharing and reuse. Thus an ontology is a specification used for making ontological commitments, which are agreements to use a vocabulary in a way that is consistent with respect to the theory specified by an ontology. Committing to an ontology enables a group of people to communicate about a domain of discourse without necessarily sharing 'true, absolute knowledge:'

> "Pragmatically, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base; each knows things the other does not (...)."
>
> (Gruber, 1993)

Closely related to the ontology of a domain is the way in which it is subdivided into subsections, often called *categories.* Section 3.4.2 discusses the various ways in which categories can be ordered to form a taxonomy. In the current section, I concentrate on plain 'set inclusion' questions.

An object can be assigned to many different categories, even to overlapping ones. A dog can rightfully be called a living creature, an animal, a mammal, a canine, a quadriped, and a pet. Glass and Holyoak (1986) discern the following base types for category systems, based on cognitive psychology:

Enumeration is simply the listing of complete extension of the whole category. Enumeration depends on the ability to name the various elements of the category in a unique way.

Definition by Properties can be done using our perceptions of the object, and as such is especially suited to concrete entities. It can be used for abstract concepts as well if we first invent abstract properties for them.

Functional and Relational Definitions share the property of being mostly a thing of our mind. Functional definitions are often 'potential means of achieving a certain type of goal,' while a relational category may be defined by the relationships between its instances and some other concept(s), such as a *doctor* being 'a person who practices medicine.' Relational definitions are widely used in normal information analysis.

Artificial Categories

The tendency to classify objects encountered in the world is pervasive in most Western thought since Aristotle, who already defined concepts in terms of *genus* (the general type, 'is-a') and *differentiae* (which distinguish a concept from the other concepts with the same *genus*). Aristotle's primitives, which he called *categories*, include Substance, Quantity, Quality, Relation, Time, Position, State, Activity, and Passivity. Some of these categories can still be found in more modern category systems. Ramon Lull developed a category system in the thirteenth century which was used by Leibniz for his *Universal Characteristic* (1679). Since then, many others have devised category systems with various numbers of primitive categories.

Vossen (1995) uses the following concepts for the base noun classification, based on work from Dik (1987):

Ensemble, Set, Mass, Object, Group, Substance, ...

Aarts (1976) presents some more proposals for top level concepts, creating a binary tree:

+/− concrete, +/− living, +/− perceptible, +/− human,
+/− shape, ...

The Longman Dictionary (1987) uses the following basic semantic codes, where each concept can have combined codes:

Concrete / Abstract, Animate / Inanimate, Organic materials / Plant / Animal / Human, Solid / Liquid / Gas, ...

WordNet (Miller, 1993) uses noun categories based on several practical considerations:[7]

Act, Animal, Artifact, Attribute, Body, Cognition, Event, ...

Analytical categories share a troubling feature: their number is sheer unlimited. Sowa (1983, p.15) presents some reasons why categories based on primitives have serious weaknesses:

- No linguistic or psychological evidence has uncovered a truly universal set of primitives.
- Most languages contain families of synonyms like *glad, happy, cheerful, lighthearted, joyful*, each with a slightly different shade of meaning. But semantic markers only support either-or dichotomies.
- Semantic markers [...] only represent conjunctions of primitives.

Natural Categories

In contrast with the artificial categories, based on reason and analysis, there are classification schemes that provide us with 'natural' categories. The term *Natural Categories* refers to the categories intuitively used by people in everyday life. They are defined in terms of perceptual similarity.

For example, psychological and anthropological research into the use of color terms (a relatively simple set of categories defined along a basic perceptual dimension, the wave length of visible light) revealed that people speaking different languages and from different cultures consistently selected exactly the same colors as typical examples of *blue, red* etc. In addition, by matching these so-called *focal colors* across languages, Berlin and Kay (1969) discovered that all languages draw their basic color terms from a set of eleven elements. In English (which has all of them, unlike some other languages) these terms are *black, white, red, green, yellow, blue, brown, purple, pink, orange*, and *gray*. Intermediate colors such as *orange* appear to be exactly in between the wave lengths of, in this case, *red* and *yellow*.

There are not many concepts that have such a clear, physical property to classify them by. Many concepts however are categorized in a way that is relatively close to the color classification, by means of *prototypes*. The prototype hypothesis was motivated by the belief that the categorization of familiar objects in the environment ultimately depends on basic, bottom-up, perceptual processes. One implication of this view is that there is a *basic level* at which people naturally divide the world into alternative categories. This level maximizes both the (perceptual) *similarities* among instances of the same category and the *differences* between instances of different categories.

Berlin also did research on folk taxonomies, and found out that animal and plant names are distributed over five different levels (Berlin, 1972):

Unique beginner: plant, animal.

[7] These categories do not show up in the WordNet noun base, but are used to keep the sources manageable and group mutually referring synsets together.

Life form: tree, bush, flower.
Generic name: pine, oak, maple, elm.
Specific name: Ponderosa pine, white pine.
Varietal name: northern Ponderosa pine.

Empirical studies showed that generic names were learned first by children. Generic names are also supposed to be more useful. In most contexts, life form names and unique beginner names will be too ambiguous and more specific names will be unnecessarily specific. For a more thorough treatment of natural categories and the experimental evidence for their existence, see (Glass and Holyoak, 1986).

Levels of Abstraction

A related way to classify words by empirical research is to look at classification chains (Vossen, 1995, p.260). Vossen argues that a Lexicon can roughly be divided into three parts: *top words*, *core words*, and *leaf words*. Words that are not used to define other words are the *leaf words*. On the other extreme of the Lexicon, *top words* use other top words to define themselves and thus will cause circularities, a sure sign that they are at the meta-level. The set of top words is relatively small and can be said to contain *primitive concepts*, not definable in other terms than themselves or by agreement. Leaf words and top words together form a limited part of the Lexicon, especially when viewed from one particular domain. The bulk of the concepts are *core words*. They are generic enough to be used in definitions of other words, but are not on the meta level.

This coincides with another finding, which usually is called the *basic level*:

> We believe that the basic level of classification, the primary level at which 'cuts' are made in the environment, [...] is the most general and inclusive level at which categories are still able to delineate real-world correlational structures.
>
> (Rosch, 1977)

Intuitively, there must be a set of concepts that the majority of people agrees upon—otherwise, the mass media would have a very hard time reaching all those millions of people. Rosch and others report extensive experimental evidence for the existence of the basic level (Vossen, 1995, p.104). This level therefore might be the best place to start at when previously unrelated groups of people need to communicate.

Robinson and Bannon introduced the term *semantic community* to refer to a group of people who live in the same environment:

> "Different groups, professions, and subcultures embody different perspectives. They communicate in different 'jargon.' Much of this cannot be translated in a satisfactory way into terms used by other groups, since it reflects a different way of acting in the world (a different ontology and epistemology)."
>
> (Robinson and Bannon, 1991)

It still is not clear exactly where to draw borders between semantic communities. Some authors provide semi-empirical ways of drawing terminology domain borders. For example, Wiederhold stresses the importance of isolating domains from each other, in such a way that within one domain all the terminology is unambiguous, both in definition and in scope, and none of the people inside the domain will have any difficulties in understanding the various words. He defines a domain to be:

> "... a subset of knowledge in the world that can be managed by a single person, or at least by a small coherent group. No compromises should be necessary to define terms, and no committee effort should be required. It is best if there is an organization willing to take responsibility for a domain ..."
>
> (Wiederhold, 1995, p.6)

But despite Wiederhold's tight definition of a domain, absolute unambiguity is almost never possible, not even among a very small group of people. However, most ambiguities in daily communication will be resolved immediately by context or by 'mental model backtracking' of the recipient (Robinson and Bannon, 1991; Ulijn and Strother, 1995).

Terminology Acquisition and Ontological Drift

I assume that two groups, who work in the same field but have no contact, will develop ontologies that might be functionally comparable, especially given Rosch's notion of basic level (the *signifiés* have a high probability of being alike) but are different in appearance (they have dissimilar *signifiants*). When contact is eventually established, the groups will need to learn each other's terminology in order to cooperate, and run into terminology clashes. This learning process is a normal phenomenon than can be facilitated by a proper Lexicon (or even by a simple data dictionary), but it cannot be avoided altogether. The specialized literature about Language Acquisition contains much information, examples, and empirical research about word acquisition. For a standard work on this theme, see Ingram (1989).

Learning each other's terminology is a first step in terminology *integration*, where two functionally equivalent sets with different appearance merge and one consistent appearance remains. When the two groups agree that their terminology should be integrated, they have to decide which words to keep and which to abandon. Neither group will favor rejection of its terminology set, so a conflict is almost certain to arise. The only way to solve these conflicts is by *agreement*, either by:

1. collectively preferring the word as used by one group and adjusting the terminology of the other group,
2. forming a committee and establishing a new, broader term,
3. creating a new term for an existing term,
4. proclaiming a total terminology overhaul.

When two *unrelated* semantic communities meet, they both face a challenge. In the most extreme case, two communities have so little in common that they cannot communicate in any way. Viewed in this light, the attempts to convey information to possible extraterrestrial life forms about humanity and the Earth's position in the Universe by means of Pioneer's and Voyager's encoded messages (Sagan et al., 1978) is a vain enterprise.

Unrelated communities must probe each other's ontology and try to construct a mental model of their partner's universe in order to even *begin* to communicate. This process of mutual trial and error is not significantly different from language acquisition within the same semantic field, but considerably more difficult given the fact that less concepts are already shared between both communities. In the context of information system development, with various communities of people involved that 'live' in semantic fields that do not overlap completely, differences in ontologies between groups lead to a phenomenon that is called *ontological drift*. Robinson and Bannon define ontological drift as:

> "the shift in meaning that can occur when knowledge artifacts (maps, designs, models) move between semantic communities."
>
> (Robinson and Bannon, 1991)

Any project that embraces several semantic communities will suffer to some extent from ontological drift. But particularly strongly phased projects will suffer more, since the phasing often is done to split up the work between specialist groups. As Robinson and Bannon say:

> "Software creation almost invariably happens *between*, rather than within, semantic communities (office workers, managers, analysts, designers, programmers, customers). [...] The interpretation of one community becomes an object for interpretation by another—but the original object of interpretation is lost in transit. There is nothing to refer back to. The original interpretation has been established as part of a different 'reality' (object of interpretation) in the new group."

Due to the nature of the involved semantic communities, the ontological drift in IS development cannot be avoided (this would mean e.g. training the users to become software engineers). But the drift should be made as explicit as possible, in order to avoid unrecognized terminology clashes that might prove expensive in a later stage of the project. The Lexicon Management System provides tools to maintain this clear separation of domain terminology.

Homonyms and False Cognates

A special form of terminology clash has to do with homonyms and misleading or false cognates. Homonyms are words with the same orthography but different meaning, where both words are used *within the same domain*. Only context analysis can solve homonym confusion. Between different domains or languages, there can be no homonyms—homonyms are by definition restricted to one single domain (Wiederhold would suggest to split the domain in such a way that it prevents homonyms

Table 3.1: Some examples of false cognates

Dutch *kaart*	=	English *card* or *map*
English *map*	=	Dutch *kaart*
Dutch *map*	=	English *folder*
Dutch *folder*	=	English *leaflet*
French *pétrole*	=	English *oil*
British English *petrol*	=	American English *gas*
	=	French *essence*
Dutch *meer*	=	German *See*
Dutch *zee*	=	German *Meer*

altogether, but I think this is not realistic, and unnecessary). Still, there can be quite suggestive overlap in orthography that easily confuses the reader. These word forms are called *false cognates*. Some examples can be found in Ulijn and Strother (1995, p.106), see Table 3.1. Within the same language, even simple words like *block* can have multiple meanings, although within the same domain they might be unambiguous. However, domain crossings are inherent in information system development, so the problem of cross-domain homonyms and false cognates should be dealt with appropriately. In my opinion, the Lexicon should make the differences between domains clear and leave the actual integration to the people. Giving a group of 'ontology students' an organized[8] Lexicon to study instead of a list of words would in any case facilitate the learning process. On top of this, structuring one's own domain often leads to more insight, even for a true specialist.

Key word in all Lexicon management therefore is *structure*. In the next paragraph, I will consider various ways in which this structure can be created and stored.

3.4.2 Concept Taxonomies

A taxonomy according to the Oxford English Dictionary is *a classification, especially in relation to its general laws or principles, [...] especially the systematic classification of living organisms*. Typical of most taxonomies is a tree-like structure and a top-down classification of supertypes into several subtypes, such that every subtype 'is a kind of' its supertype.

Specialization and Generalization

Not all super/subtype classification schemes follow the same basic classification strategy. The most significant difference in classification is that between *specialization* and *generalization*. In a specializing taxonomy, each subtype represents a more specific type than the supertype, while in a generalizing taxonomy, the subtype represents a more generic concept.

[8]Not organized in the sense of 'alphabetically,' but with semantic links between the various words.

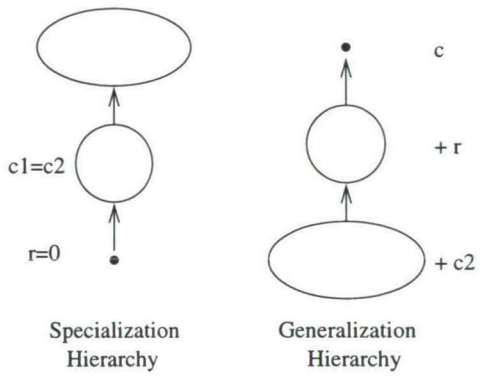

Figure 3.4: Specialization versus Generalization Hierarchies

Typical of the specialization scheme is that only the 'leaf nodes' at the very bottom of the hierarchy represent true existing entities: lions exist, but felines or mammals do not.[9] Generalizing taxonomies take specific instances in a world and then subclass those instances to cover more generic cases. In a generalizing taxonomy, all classes can have instances. A property of both specialization and generalization is the fact that *class migration* is prohibited, i.e., a lion cannot change into a cat, despite the fact that both species are feline. If we have a feline, it must necessarily be either a lion or a cat from the very moment it comes into existence. Whenever class migration appears, another type of hierarchy should be considered, see the section about role playing on page 44.

As an example (Figure 3.4), a circle and an ellipse can both be called each other's subtype. The standard mathematical ('analytical') view prefers a specializing taxonomy: a circle is an ellipse (follow the arrow) with a special, restricting property $c_1 = c_2$. But the 'programming' view prefers a generalizing taxonomy: an ellipse can be considered to be a circle with an extra 'center,' monotonically extending the object's attributes. OO programming textbooks often give this kind of examples; another one would be COORDINATE ← POINT ← SQUARE ← LINE/RECTANGLE. There is a degree of arbitrariness involved in the creation of an attribute inheritance tree.

Specializing taxonomies are often preferred because they offer a more analytical view of the world. But generalizing taxonomies have the advantage of a more bottom-up approach and support gradual exploration of a new field without the continuous need for extensive taxonomy revision.

Knowledge Acquisition

Generalization is a fundamental learning strategy. Confrontation with individual cases intuitively leads to assumptions about other, apparently similar cases. It is this generalization mechanism that gives us some ability to predict the future and

[9] We have learned to see lions as special cases of felines, and therefore accept the existence of felines as a category. But the class of felines does not have instances of its own; it only has subclasses (a class such as *feline* is commonly called an *abstract class*).

allows us to control our environment to a certain extent. But it is a strategy, not an analytical solution. Sometimes, people's predictions based on generalization are wrong, probably for one of two main reasons: (Littlewood, 1984, p.23)

1. For some reason, the rule does not apply to a particular item, even though it has been allocated to the appropriate category. Therefore an *exception* to the general rule must be learned.

2. The item belongs to a different category, which is covered by another rule. Therefore either the item must be reallocated to a different category, or an entirely *new category and rule* must be constructed.

In either case, the initial error was due to *over*-generalization of the rule which caused the wrong prediction.

A Lexicon should support acquisition strategies based on generalization, and therefore also support over-generalization and the subsequent correction. It would be nice to *prevent* over-generalization altogether, but the essence of building a Lexicon is that the necessary information is not yet completely available.

Support for generalization alone is not enough. It may happen that some analyst closely inspects a certain class of objects, and discovers some interesting variation within the class that has remained unexplored until then. Or some new theoretical insights offer a way of organizing the ontology in a less intuitive, but analytically more interesting way. These analytical approaches to knowledge acquisition are as valuable as 'empirical' ones, and they produce a *specialization* taxonomy based on hyponymy relationships.

Hyponymy relationships are typical for Lexicons that are created by thoughtful construction. Calzolari even considers them the norm:

> "Hyponymy is the most important relation to be evidenced in a Lexicon. Due to its taxonomic nature, it gives the Lexicon, when implemented, a particular hierarchical structure: its result is obviously not a tree, but many tangled hierarchies."
>
> (Calzolari, 1993b, p.171)

I do not readily agree with her statement that the analytical specialization hierarchy is the most important one. In my opinion, both specialization and generalization hierarchies are equally important, and both can greatly help understanding a particular domain. Moreover, existing Lexicons such as WordNet (Miller et al., 1993) also offer antonymy, synonymy, hypernymy, and other relations, that can all significantly help to create a tight semantic net around a concept.

Role Playing and Aggregation

Contrary to generalization and specialization hierarchies, some taxonomies *do* allow instances to migrate between classes. For example, when a person starts a study, she becomes a student and thus gains properties and attributes. Common sense suggests that a student 'is a' person, and often this is modeled as a true class switch from person to student. This involves either a lot of low-level attribute copying or

a link between a 'person' and a 'student' instance—a sure sign that something is wrong. What is wrong is that the student is now viewed as the general case of the person, the classic OO programming approach.

What really happens is not a fundamental change of the instance (an observer will not see any change in the girl), but the instance taking on a new *role*. Roles offer access to new actions that the actor can perform, but do not change the actor. Consequently, the same actor can play many roles, even at the same time, and can usually choose which roles to play and when. Viewed this way, roles have much more in common with sets than with type hierarchies.

Sowa makes this same distinction when he talks about types (generalization/specialization) and sets (arbitrary collections of objects with some common property, such as being a student). He says:

> "Many people confuse types and sets. Yet there is a fundamental difference. Statements about types are *analytic*; they must be true by intension. Statements about sets are *synthetic*; they are verified by observing the extensions. To say that the intersection of the set δCAT[10] with the set δDOG is empty, or $\delta\text{DOG} \cap \delta\text{CAT} = \emptyset$, simply means that at the moment no individual happens to be both a dog and a cat. [...] But to say that $\text{DOG} \cap \text{CAT} = \perp$[11] means that it is logically impossible for an entity to be both a dog and a cat."
>
> (Sowa, 1983, p.82).

Besides roles, there are other hierarchical relationships between concepts that allow class migration. One of the most salient is the aggregation hierarchy, which links concepts in terms of 'part-of' semantics (Storey, 1991; Gupta and Sykes, 1996).

Whereas a specialization hierarchy might rightfully be paraphrased as 'any x is a kind of y,' an aggregation hierarchy would need to be paraphrased as 'any element (part) of the set of x-es also is an element (part) of the set of y-s.' This expression does *not* imply any 'is a kind of' semantics, despite the human tendency to interpret the expression this way.

As an example, see Figure 3.5. A CAR contains an ENGINE, which contains PISTONS. Although a tree might look appropriate at first sight, it is not the case that a PISTON 'is a kind of' ENGINE, let alone 'a kind of' CAR. However, a set representation (carrying the 'part of' semantics) would be correct. Aggregation therefore should be represented by Venn diagrams instead of trees, not because a tree would be inappropriate, but because it suggests the wrong kind of semantics. It should also be noted that aggregation hierarchies do not provide property inheritance at all.

Positioning of Concepts in the Taxonomy

When a new concept needs to be included in the Lexicon, the lexicographer must search for an existing concept to attach the new concept to, thus maximizing the

[10]The δ symbol is pronounced as 'denotation of.' δCONCEPT is the set of all entities that are instances of CONCEPT.

[11]The 'absurd' symbol $\perp$ in this case means that there is no (maximal) common subtype.

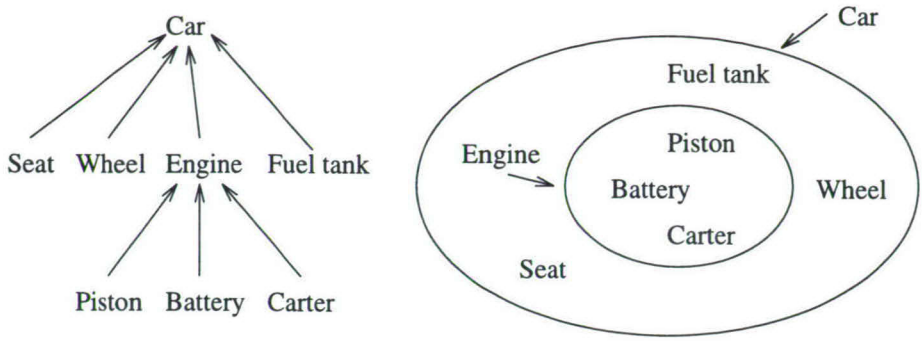

Figure 3.5: Aggregation tree versus Venn diagram

amount of reused information. On the other hand, people may spend time restructuring the hierarchy, working their way from the top down. They have an overview of the whole Lexicon and can decide to move concepts around to fit them together. The resulting structure will tend to be rather fixed at the abstract top and become less and less stable further down the Lexicon, where there is still no 'ideal' structure.

When I look at the Lexicon on the three mentioned levels (see Section 3.4.1), I can see some parallels between the various ways of splitting the Lexicon in three parts.

1. Top level: abstract words, fixed according to theory, highly static;
2. Middle level: generic words, often acquired by MRD,[12] moderately static;
3. Bottom level: domain terminology, in process of being acquired, highly dynamic.

It is important to notice that each level can exist without the support of the other levels. In particular, the domain terminology can be stored effectively without any common words—but this will not be efficient, since every domain concept will need to be connected to all related domain concepts by hand. The three-layer architecture will enable us to work with the Lexicon on three separate levels (theoretical, corpus/statistical, and practical) without requiring that e.g. the theoretical level has been fixed before we can attach the MRD. The more work goes into the upper two layers, the fastest we can add new domain terminology, but it is not an absolute requirement.

3.4.3 Frames

Much Lexicon information is not stored in tree or set structures, but in tuples that are traditionally called *frames* (Section 3.2.1). The way in which the Lexicon treats frames is almost equivalent with Sowa's Conceptual Graphs (Sowa, 1983), mainly because all concepts in the Lexicon are free-floating objects with interconnecting

[12]Machine Readable Dictionary

links and thus are well suited to a graph approach with nodes and edges. I will briefly present Conceptual Graph theory in the next sections.

Concepts

The 'concept' part of the CG lexicon contains an inheritance hierarchy with supertypes and subtypes, and many of the concepts (especially those that represent actions, attributes, and role types) are always attached to certain types of relations. For those concepts, Sowa lists a canonical graph that defines selectional constraints on possible combinations of concepts. These graphs are not definitions, but more a kind of prototypical example of usage. Additionally, he provides a short English sentence to help the reader to relate the concept to the semantic network in his or her own head. Examples of concepts are:

ACT < EVENT. An act is an event with an animate agent.
[ACT]→(AGNT)→[ANIMATE].

AGE < CHARACTERISTIC. Age is characteristic of an entity at a point in time.
[AGE]-
(CHRC)→[ENTITY]
(PTIM)→[TIME].

ANGEL < ANIMATE, MOBILE-ENTITY, ¬PHYSOBJ. An angel is an animate being, but not an animal.

ARRIVE < ACT. A mobile entity arrives at a place.
[ARRIVE]-
(AGNT)→[MOBILE-ENTITY]
(LOC)→[PLACE].

CUT < ACT. An animate being cuts a physical object with another physical object that has attribute sharp.
[CUT]-
(AGNT)→[ANIMATE]
(INST)→[PHYSOPBJ]→(ATTR)→[SHARP]
(OBJ)→[PHYSOBJ].

Conceptual Relations

The 'relation' part of the lexicon contains relations between concepts. Again, these entries are not definitions, but constraints on the use of the relations in conceptual graphs. Examples are:

accompaniment. (ACCM) links [ENTITY:*x] to [ENTITY:*y], where *y is accompanying *x. Example: *Ronnie left with Nancy.*
[LEAVE]→(AGNT)→[PERSON:Ronnie]→(ACCM)→[PERSON:Nancy].

agent. (AGNT) links [ACT] to [ANIMATE], where the ANIMATE concept represents the actor of the action. Example: *Eve bit an apple.*

[PERSON:Eve]←(AGNT)←[BITE]→(OBJ)→[APPLE].

attribute. (ATTR) links [ENTITY:*x] to [ENTITY:*y] where *x has an attribute *y. Example: *The rose is red.*

[ROSE: #]→(ATTR)→[RED].

method. (METH) links an [ACT:*x] to a [SITUATION:*y] that shows how the act *x is accomplished. Example: *Larry caught the crook with a mighty leap.*

[ACT: [PERSON: Larry=*x]←(AGNT)←[CATCH]→(OBJ)→[CROOK]]-
(METH)→[ACT:[PERSON:*x]←(AGNT)←[LEAP]→(MANR)→[MIGHTY]].

Conceptual Graph theory emphasizes relational aspects between various concepts. The main information that can be found in the conceptual part of the lexicon is the generic subsumption hierarchy, together with conceptual graphs that constrain the possible combinations of concepts. The relational part lists the possible link types between concepts and contributes more selectional constraints (usually on concepts higher up in the subsumption hierarchy).

When compared to FG, the CG relations coincide with FG roles, and the CG concepts equate FG frames, where entity (noun) types are considered to be frames as well (Mackenzie, 1987). Both theories differ in the amount of detail they include; Conceptual Graphs have a much more elaborate way of defining the exact properties, constraints, and even usage of roles than does FG. In fact, CG leaves this definition to the analyst/lexicographer while FG treats them as part of the theory. On the other hand, FG is much more powerful in capturing linguistic details, partially because FG's aim was to be a good linguistic theory; CG is a knowledge representation theory.

3.4.4 Terminology Domains

In order to enable a Lexicon to support more than one project at the same time, there must be a mechanism to separate the terminology of each project from the other project(s). If this is not properly done, unnecessary terminology clashes between the two 'domains' will occur. Common terminology (usually existing in the Lexicon before either project was started) should be given its own domain as well.

Analogous to DBMSses, a Lexicon Management System (LMS) should separate the various users' definitions of new concepts and offer the ability to 'publish' these definitions to share them with other users or groups of users later on. I propose to follow the 'pessimistic' view on resource sharing and security that has become commonplace in the database and systems engineering world. In this view, concepts are not shared at all by default—they are only visible to the particular LMS user that entered them. Any concept that is introduced into the LMS will need to be explicitly cleared for common usage in a specific domain. This requires some extra work, best described by *domain editing*, that should be done by knowledgeable domain specialists.

Unlike specialized table definitions and their contents in a DBMS, Lexicon concept definitions might be shared in many more ways than by just granting access to other

LMS users. Concepts that have the same lexical appearance but mean different things in different domains might hold a key to successful prediction of domain integration problems. The same concept can turn out to be used in more domains, possibly using different lexicals, and can thus be used to bridge the domain gap. Some domains may use a more specialized version of the same concept. Analysts may introduce abstract domains, not existing in reality but of great value to gain theoretical insight. These abstract domains can use inheritance as a powerful mechanism for classification and re-use of existing terminology, building taxonomies in the process.

Domain Management

During terminology clash resolving, it is of paramount importance to separate the terminology sets of both groups, but it must be possible to use both sets together, e.g., to compare conflicting terms or to mark one term as the preferred one. When working with one terminology set at the time, each term may be presented without any further specification as long as the terminology set name (such as 'manufacturing') is in one way or another visibly present. When two or more domains are combined, we need a way to clearly distinguish terms that have the same appearance, yet different meaning. Furthermore, these distinctions should be presentable in plain textual format, not by elaborate computer screen mark-up (with colors or graphics).

These requirements suggest a type of *prefix naming* that can also be found in various other domain structures that are known in the worlds of science and technology. Well-known examples are SQL table names, that by default are prefixed with a unique user identification to solve possible duplicate table names, and Internet domain names (Mockapetris, 1983), where every domain has a central naming organization that is responsible for all the names in the particular domain.[13] By prefixing the word with the domain name ('personnel.employees' versus 'payroll.employees') when crossing domain boundaries, it is possible to keep using both terms, yet to avoid the conflicts inherent in cross-domain terminology, or to make them explicit. To minimize the burden for the intra-domain Lexicon users, it seems appropriate to include the notion of a 'default domain' that gets prefixed before every word that has no explicit domain specifier.

Domain structures[14] are typically built bottom-up if related domains appear to have some (partial) terms in common. It is highly unlikely that the domain hierarchy will parallel the basic term hierarchy.

Although the LMS does no force the user to use a specific tree, it might prove useful to organize the top levels of the domain tree in a standard way, because this will enable various applications to retrieve the necessary information in a consistent way across various Lexicons or even between various LMSses. This common domain tree is much more valuable than e.g. a common concept tree, because domains can be used to store many essential properties of and relations between concepts, whereas the concept tree only represents property *inheritance*.

[13]Internet domains are not prefixed but postfixed, and naming organizations can give a mandate to sub-naming organizations.

[14]Domains organized in taxonomies; these should not be mixed up with the taxonomy of the terminology *in* a domain.

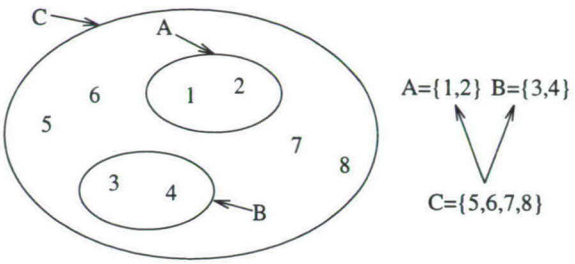

Figure 3.6: An example of subset representation through inheritance

Subsets through Inheritance

Terminology domains are built in the Lexicon by means of a generic mechanism, that of the Set. Sets are specialized Lexicon objects that can hold references to other objects, but being objects themselves, they participate in the normal inheritance hierarchies.

(References to) concepts can be placed in an unrestricted number of sets, and it is possible to draw Venn diagrams that represent the overlap and separation of domains and the concepts in them. Because sets cannot be included in other sets by themselves (the domain of a set is restricted to lexicals, frames, and linkers),[15] inheritance must be used to provide the equivalent of subsets.

Given three sets $\mathcal{A}, \mathcal{B}, \mathcal{C}$ with $\mathcal{A} \subset \mathcal{C} \wedge \mathcal{B} \subset \mathcal{C}$, this can be expressed in Lexicon terms by making set $\mathcal{C}$ a subtype of both set $\mathcal{A}$ and $\mathcal{B}$ and deleting $\mathcal{A} \cup \mathcal{B}$ from $\mathcal{C}$ (see Figure 3.6). This might seem counterintuitive at first sight, but can be understood as follows. Inheritance between sets is on the *sets*, not on their *elements*. Therefore, every element in a parent set is 'added' to the elements in the child set. In the example, set $\mathcal{C}$ only contains the elements $\notin \mathcal{A} \cup \mathcal{B}$—yet when queried, $\mathcal{C}$ will report to contain {1,2,3,4,5,6,7,8}, or $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$.

Analogously, when $\mathcal{B}$ and $\mathcal{C}$ are both children of $\mathcal{A}$, it holds that after inheritance processing $\mathcal{B} \cap \mathcal{C} = \mathcal{A}$.

A formal definition of the subset-subtype relation can be found in Section 4.2.6.

3.4.5 Lightweight Domains

Antonym sets are an example of the usage of the domain mechanism to store typically binary relationships. When there are two concepts, say, SMALL and LARGE, they can both be made member of a domain. When this domain is placed in the domain hierarchy under the (abstract) domain named 'antonyms', lexical software can infer that the elements of the domain are each other's antonyms. The antonym set itself does not need to have a lexical, because its existence only already conveys the intended purpose: to indicate that its two members are antonyms.

As observed by Miller (1993), antonym relations are not between concepts but between lexicals (see Figure 3.7). This means that typically, concepts will be placed in synonym sets and their associated lexicals in antonym sets. Such a structure

[15] More general, to Level 3 objects—see Section 4.2.6.

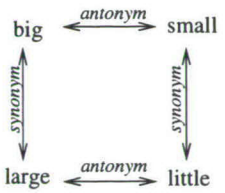

Figure 3.7: Synonyms versus antonyms

enables the lexicographer to store synonym relations (big, large) and (little, small) and antonym relations (big, little) and (large, small) in such a way that various paths can be taken through the lexical database to find the proper word forms, e.g., when all the synonyms of the antonym of SMALL are searched for.

Another important feature for Lexicon organization is the *contrast set* (Weigand, 1990, p.102). These sets of somehow related words, such as {*Sunday, Monday, ..., Saturday*} or {*mother, father*}, often with an appropriate cover term (*day, parent*), offer powerful extensions to the taxonomy—in fact, contrast sets themselves can be grouped in another taxonomy, leading to semantic fields.

Both types of sets cannot properly be specified by using the main LMS Set mechanism, reason why I also provide for 'lightweight' domains that can be created by the lexicographers themselves as they see fit.

Chapter 4

The Lexicon Management System

— in which the writer assembles the toolkit he needs to properly describe and manipulate lexical structures, followed by a detailed description of the way he would design an LMS *in case anyone should ask him to —*

In this chapter, I present the internal structure of the Lexicon Management System (LMS), i.e. the Lexicon without the actual contents.[1] First I give a general overview of the basic structure, to give the reader some sense of what the LMS looks like. Then I present a formal definition of all the underlying concepts, leading to a design of a Lexicon Management System that can be implemented. The chapter is concluded by a formal description of the proposed command language for the LMS, called LIX, and some example LIX sessions.

4.1 Overall Structure

This section describes the overall structure of the Lexicon Management System, emphasizing the layered design and the specific purposes of each layer.

The LMS consists of three main layers, which are called the *storage layer*, the *paradigm/language layer*, and the *lexicographic layer*. For easy reference in the rest of this chapter, I will designate these layers by their rank number (1, 2, 3), often prefixed with an '@' sign (pronounced 'at'). Technically, there is a 'zero' layer as well, the machine level, which will come into play as soon as an actual implementation on a computing engine is created. Although some design considerations at this low level have been incorporated into the LMS design, I will not cover them extensively in this thesis.

4.1.1 The Storage Layer

The Storage Layer (1) is the only truly fixed part of the LMS, comparable to the database management software of a traditional information system. How this storage layer is actually implemented (@0) is platform-dependent, but the layer 1 interface to layer 2 is fixed and independent of the LMS application, domain, language, or linguistic paradigm used.

[1] A Lexicon Management System and a Lexicon relate to each other like a Database Management System and a Database.

At layer 1, the LMS offers a limited range of relatively primitive data structures that are tuned towards the storage and retrieval of lexicographic data. The storage layer is not meant to be used by the lexicographers who actually fill the lexicon, nor by the users of the filled Lexicon. Only lexicologists and linguists who intend to create a new Lexicon should have knowledge of layer 1, so that they can create the necessary basic elements to fit their paradigms and morphosyntax of choice.

Technically, only layer 1 objects can exist in any LMS implementation, and layer 2 and 3 objects are always mapped to layer 1 objects. However, much of this mapping is completely transparent to the LMS users.

Typical layer 1 objects are tuples, regular expressions, strings, and lexicals.

4.1.2 The Paradigm/Language Layer

Layer 2 designates the theoretical paradigm and linguistic rules of the Lexicon, which approximately equal the database schema of normal data bases. Layer 2 is not fixed during the design and implementation of the LMS (actually layer 1), but should be constructed once during the lifetime of a Lexicon. It provides the lexicographers (@3) with a stable base that uses their own terminology to store lexicographic data, and can check for some consistency and other rules during Lexicon updates. In principle, the same LMS can host several paradigm layers, although the current model offers no features to separate their presentation to the lexicographers.

The Paradigm/Language layer is split into two parts. The Paradigm part contains the basic (lexical) objects of a linguistic theory, such as Functional Grammar, which generally are used to store semantic information and usage rules. The Language part holds the information needed for morphosyntactic language-dependent queries at the word (and sometimes phrase) level. It is feasible to have one paradigm and several languages @2, which offers the possibility to cross-link languages when desired.

Domain-dependent information, such as a feature that indicates the preferred word for synonym sets or other pragmatic features that are not bound to any linguistic paradigm, is also stored in the paradigm part @2. The reason for this is that a Lexicon can only maintain consistency when basic features are centrally defined and the users cannot just add them at will. Layer 2 then becomes the natural place to store this type of information. From a lexicographer's point of view, there is little difference between a linguistic theoretical framework and pragmatic features.

A typical paradigm would require objects such as terms, frames, and roles, possibly extended with predicate formation rules. Languages could have nouns, verbs, prepositions, and noun phrase formation rules. Pragmatic features could be 'is preferred word' and 'belongs to a particular terminology domain.' All these objects @2 are fully configurable by the lexicologist/linguist in charge of creating the paradigm and language layer.

4.1.3 The Lexicographic Layer

Layer 3 holds the actual Lexicon contents (that is, the contents is presented @3; it is actually stored @1). Using only the predefined objects available @2, the lexicographers build up a network of interconnected objects @3 that represents the lexical

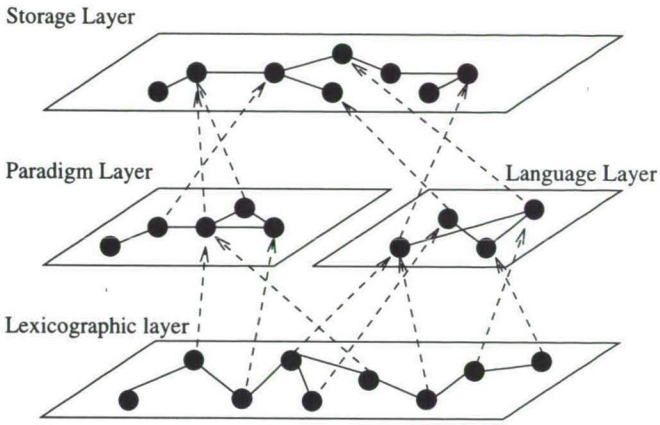

Figure 4.1: A 3D Representation of the LMS

part of specific domain knowledge.

The particular design model of the LMS causes most information @3 to be distributed over several objects @1 which are linked together as required by the rules @2. As such, the Lexicon resembles both a network database and an object-oriented database. This causes difficulties in querying and updating, because the LMS at its most basic layer (1) cannot deal with e.g. complex objects. Some intermediate engine should remap complex objects to interconnected simple objects. Most of the information necessary for such a remapping is available @2, but practical circumstances will often require more specific user interfaces than a generic LMS can provide. Just as with normal database systems, there is a need for *applications* put on top of the LMS to ensure that the daily usage becomes practical.

4.1.4 Classes, Instances, and Inheritance

The three LMS layers together form a complex mesh of objects that have several relations. In Figure 4.1, I present a three-dimensional view of the LMS which should illustrate the generic relationships. Within each layer, all objects have parent-child relationships with multiple inheritance. At layer 1, these relationships are fixed and the inheritance hierarchy merely is a convenient way to simplify the conceptual representation; some LMS implementations will provide the base object types (classes) in plain non-OO code. Layers 2 and 3, however, are user-defined and here, inheritance can successfully be used to save work and clarify possibly complex structures. In general, inheritance is a feature of the LMS that can and should be used when appropriate, but it is not at all mandatory.

The parent-child inheritance chain (solid lines in Figure 4.1) is the only standard LMS relationship that exists *within* a layer. Class-instance relationships (dashed lines) are only possible *between* layers.

As a general rule, the 'base classes' at layer 1 are the only true LMS classes in the OO sense, i.e. they can be instantiated into discrete objects with multiple, equally structured objects per class. All instances have unique object identifiers, a certain

state (represented as values of the attributes provided by their class), and often some operators (methods) that are specific for their class.

At layer 2, which in effect is a specification layer, most objects are simple strings together with a pointer to a base class @1. The string provides just a reference name, given by the linguist who creates the theoretical paradigm and the language morphosyntax. The users who work @3 can use these familiar strings to work with the paradigm and language terminology instead of with the technical layer 1 terminology. However, some complex classes @1 need more than a single string @2 to be of use to the LMS users. For example, Lexicals need a set of rules @2 to provide for regular morphosyntactic word forms, and Sets keep their extension @2 as well. LMS users view all objects @2 (with the exception of the Sets) as true OO classes that can be instantiated @3, even while this is not at all the case in reality.

Layer 3 holds the true Lexicon contents, as a collection of objects that are all instances of 'classes' @2, that refer to each other by means of their object identifiers, and that can be placed in Sets @2 to indicate specific features. In this layer, no new classes or Sets can be created, but there is no limit to the number of instances of Concepts, Tuples, and Lexicals.

In the next section, I will extensively discuss the various base classes @1. Later in this thesis (Section 4.3), the application of these base classes @2 is explained and examples are given of an actual linguistic paradigm and language implementation in the LMS.

4.2 LMS Base Classes

This section presents the various Base Classes (LMS layer 1) that make up the primitives out of which any Lexicon schema must be constructed. Because layer 1 is a rather technical part of the LMS, I choose to keep the descriptions of the Base Classes tight and to the point, without examples. Where necessary, formal definitions will be used to specify exact features and behavior. The Base Classes are presented in approximate order of complexity.

The LMS Base Classes are: Tuple (class and instance), Alpha-numerical string, Regular expression, Set (class and instance), and Lexical (class and instance). These classes will be defined and explained in detail in Sections 4.2.5 to 4.2.7; some generic definitions are given here.

4.2.1 Objects

Definition 1

oid is the set of object identifiers.

The set of Base Classes $\Omega_1 =$
$\{\mathsf{TupCls}, \mathsf{TupIns}, \mathsf{String}, \mathsf{RegExp}, \mathsf{SetCls}, \mathsf{SetIns}, \mathsf{LexCls}, \mathsf{LexIns}\}$

The set of all objects Ω consists of objects o, with each $o \in \Omega = \langle i, b \rangle$ where $i \in$ **oid**, $b \in \Omega_1$.

□

Object identifiers are usually simple integers $\in \mathbb{N}$. They have no mathematical operations such as addition or multiplication defined on them, and thus have no ordering; they are solely used for identification.

Definition 2

Given two objects $o_1, o_2 \in \Omega$, $o_1 = \langle i_1, b_1 \rangle$, $o_2 = \langle i_2, b_2 \rangle$. Then $o_1 = o_2 \iff i_1 = i_2$

□

Even if two objects $\in \Omega$ are instances of the same Base Class and have completely equal attribute functions but different object identifiers, they are considered unequal.

For consistency and easy reference, I will split Ω into two non-overlapping sets:

Definition 3

$\Omega = \Omega_2 \cup \Omega_3$
$\Omega_2 \cap \Omega_3 = \emptyset$

□

Ω_2 contains all objects @2 (Lexicon classes), Ω_3 all objects @3 (Lexicon instances). The Lexicon Base Classes are not included in Ω (and therefore have no **oids**), but I call this set Ω_1 nonetheless to get a straightforward equivalence between layers and object/class sets.

4.2.2 Lexicon Architecture

I define the Alphabet $\mathcal{A}$ of the Lexicon as a set of glyphs in any required orthography, including diacritics. In this thesis, I assume that $\mathcal{A} = \{A..Z,\ a..z\}$ for simplicity.

The following definition sets up the Lexicon architecture, with Lexicon classes @2 and Lexicon instances @3 (see Figure 4.1).

Definition 4

A string s is a list of characters $c_1, \ldots, c_n \in \mathcal{A}$, with $n \geq 1$.

For each object $o \in \Omega_2$, there is a string s such that $\mathbf{name}(o) = s$.

A Lexicon Definition $\mathcal{D}$ is a set of class objects, such that
$\mathcal{D} \subset \Omega_2 \ \wedge$
$\forall o_1, o_2 \in \mathcal{D}: \ \mathbf{name}(o_1) = \mathbf{name}(o_2) \iff o_1 = o_2 \ \wedge$
$\forall o \in \mathcal{D}: \ \mathbf{parent}(o) \subset \mathcal{D} \setminus o$

A Lexicon $\mathcal{L}$ is a set of instance objects, such that for each $\delta \in \mathcal{D}$:
$\mathcal{L} \subset \Omega_3 \ \wedge$
$\forall o \in \mathcal{L}: \ \mathbf{class}(o) \in \delta \ \wedge$
$\forall o \in \mathcal{L}: \ \mathbf{parent}(o) \subset \mathcal{L} \setminus o$

□

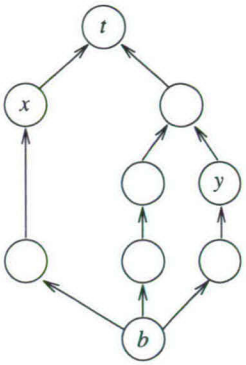

Figure 4.2: Three Parallel Ancestor Chains

4.2.3 Inheritance

indexinheritance!definition

As explained in Section 4.1.4, there is no traditional inheritance between classes @1, but the *instances* of these classes at levels 2 and 3 can have their own hierarchy if the LMS end users decide to create one. Note that this inheritance is only allowed between objects at the same level; between levels, there is no inheritance. This restriction is already implied by Definition 4, as is the requirement that an object is not its own parent. I now define inheritance by defining a partial ordering on Ω, as inspired by Weigand (1990):

Definition 5

For all $o_1, o_2, o_3 \in \Omega$, it holds that:

$o_1 \vdash^1 o_2 \iff o_2 \in \mathbf{parent}(o_1)$
$o_1 \vdash^1 o_2 \ \wedge \ o_2 \vdash^1 o_3 \ \rightarrow \ o_1 \vdash o_3$
$o_1 \vdash o_2 \ \wedge \ o_2 \vdash o_3 \ \rightarrow \ o_1 \vdash o_3$
$o_1 \vdash o_2 \ \rightarrow \ o_2 \nvdash o_1$
$o_1 \vdash \Delta$
$\nabla \vdash o_1$

□

The $\vdash^1$ operator is pronounced as 'is child of,' the $\vdash$ as 'is descendant of.' The set of ancestors of an object (including the top element Δ) is called the *heritage* of that object. The ordered chain from an object up to Δ is the object's *ancestor chain.*

The main aim of the inheritance mechanism is to provide for simple default inheritance. Usually this means that simple values will be overridden by children, completely replacing the value they might inherit from their parent(s), and that complex values such as sets will be extended.

Multiple Inheritance

With multiple inheritance, where objects can have more than one parent (as in Figure 4.2; the (local) top node t is not essential), there are two problems to cope with.

The first problem is called *name clash* (Booch, 1994, p.123). It appears when several parallel ancestor chains each provide the same attribute, so that there is no unambiguous name resolution any more. In Figure 4.2, this would be the case if both objects x and y held the same attribute.

The second problem with multiple inheritance (Booch, 1994, p.123) is called *repeated inheritance*, where a certain attribute originates in a common ancestor of all parallel ancestor chains (t in Figure 4.2) and gets distributed down the inheritance tree to the bottom element in several distinct, and therefore possibly conflicting ways (for example if one ancestor chain redefines the attribute and the other chains do not).

To avoid these problems, I propose to check the intended heritage of a new object before it is actually created, and to reject the object creation if one of the problems occurs. The other solution, to implement various decision mechanisms to solve such problems when they are encountered, introduces a degree of arbitrariness and could negatively influence query response time.

4.2.4 Strings and Regular Expressions

Objects of the class String and RegExp are siblings. Whereas RegExps specify a set of strings by a single regular expression, Strings enumerate the complete extension of their string set.

Definition 6

Given an alphabet $\mathcal{A}$ and the associated string space $\mathbf{str}(\mathcal{A})$ that consists of all strings that can be formed by permutation and concatenation of the glyphs in $\mathcal{A}$. $\mathbf{str}(\mathcal{A})$ typically is infinite.

For each String object $o \in \Omega_2 = \langle i, b \rangle$, $i \in \mathbf{oid}$, $b =$ String, there is a function **ext** to $\mathbf{str}(\mathcal{A})$ that gives the extension set of the String object.

□

For regular expressions, the definition of the **ext** function is more restricted.

Definition 7

Given a regular expression string space $\mathbf{reg}(\mathcal{A}, r)$, consisting of all strings $\in \mathbf{str}(\mathcal{A})$ that can be formed by interpreting a string r as a regular expression on $\mathcal{A}$. I do not give a detailed presentation of the mechanism of regular expression expansion here; for more information, see Sebesta (1989).

For each RegExp object $o \in \Omega_2 = \langle i, b \rangle$, $i \in \mathbf{oid}$, $b =$ RegExp, there is a function **value** such that $\mathbf{value}(o) = r$, and $\mathbf{ext}(o) = \mathbf{reg}(\mathcal{A}, r)$ gives the extension set of the RegExp object.

□

If Strings or RegExps have parent(s), their extension is enlarged by the extension of their parent(s). The function that recursively follows the heritage of an object is called **ext***:

Definition 8

$$\mathbf{ext}*(o) = \mathbf{ext}(o) \cup \bigcup_{p \in \mathbf{parent}(o)} \mathbf{ext}*(p)$$

□

4.2.5 Tuples

Ω_1 contains two classes that together form the Tuple mechanism: TupCls and TupIns. I call the instances of TupCls *Tuple Classes* @2, and the instances of TupIns *Tuple Instances* @3. Tuple Classes are elements of Ω_2 and are part of a Lexicon Definition, while Tuple Instances are in Ω_3 and form a Lexicon according to a Lexicon Definition.

Tuple Classes

Definition 9

For each TupCls object $o \in \Omega_2 = \langle i, b \rangle$, $i \in$ **oid**, $b =$ TupCls, there is a function **tuple** to $\wp\Omega_2$ such that:

$\mathbf{tuple}(o) = \langle o_1, o_2, \ldots, o_n \rangle$, with $o_1 \ldots o_n \in \Omega_2$ and $n \leq 0$.

□

Tuple Instances

Tuple Instances @3 are always bound to a particular Tuple Class @2.

Definition 10

For each TupIns object $o \in \Omega_3 = \langle i, b \rangle$, $i \in$ **oid**, $b =$ TupIns, it holds that $\mathbf{class}(o) = o_c$, with o_c a TupCls object $\in \Omega_2$

and there is a function **tuple** to $\wp\Omega_3$ such that
$\mathbf{tuple}(o) = \langle o_1, o_2, \ldots, o_n \rangle$, with $o_1 \ldots o_n \in \Omega_3$ and $n \leq 0$.

□

The objects in the tuple returned by the **tuple** function need to be of the level 2 class as specified by the associated TupCls object.

4.2.6 Sets

All Set objects are roughly comparable to String objects, in the sense that they hold a function **ext** which returns a set. However, there are two Set classes @1: SetCls, which instances hold a set of objects $\in \Omega_3$, and SetIns, which instances are bound to a particular SetCls instance, and also hold a set of objects $\in \Omega_3$. The main practical difference between the two Set classes is that a SetCls is part of the Lexicon Definition, and thus must be created by a linguist before the Lexicon is actually built, while SetIns objects can be made by lexicographers as they see fit. Consequently, SetCls objects have names, while SetIns objects do not (Definition 4).

Set Classes

Definition 11

For each SetCls object $o \in \Omega_2 = \langle i, b \rangle$, $i \in$ **oid**, $b =$ SetCls, there is a function **selrestr** to Ω_2 which contains a selection restriction on the (level 2) class elements of o's extension.

Additionally, there is a function **ext** to Ω_3 that gives the extension set of the SetCls object:
$\mathbf{ext}(o) = \{o_1, o_2, \ldots, o_n\}$ with $\mathbf{class}(o_i) = \mathbf{selrestr}(o)$ for each i between 1 and n.

□

When a SetCls instance has parents, their extensions are added to the SetCls's own extension through the **ext*** function:

Definition 12

$$\mathbf{ext}*(o) = \mathbf{ext}(o) \cup \bigcup_{p \in \mathbf{parent}(o)} \mathbf{ext}*(p)$$

□

Set Instances

Instances of the SetIns class have almost the same properties as instances of the SetCls class, but they exist at level 3 and get their selection restriction out of the associated SetCls object.

Definition 13

For each SetIns object $o \in \Omega_3 = \langle i, b \rangle$, $i \in$ **oid**, $b =$ SetIns, there is a function **ext** to Ω_3 that gives the extension set of the SetIns object:
$\mathbf{ext}(o) = \{o_1, o_2, \ldots, o_n\}$ with $\mathbf{class}(o_i) = \mathbf{selrestr}(\mathbf{class}(o))$ for each i between 1 and n.

□

Inheritance on SetIns objects is defined in the same way as for SetCls's.

4.2.7 Lexicals

The class of Lexicals is the most specialized and most complicated Base Class of the LMS. Instead of unordered sets or position-sensitive tuples, Lexicals contain explicitly named attributes in data structures called *attribute vectors*. The following sections define the formal notions needed for these data structures.

Definition 14

Given a set K_α of attribute keys $\{\kappa_1, \kappa_2, \ldots, \kappa_n\}$, and a tuple V_α of attribute values $\langle v_1, v_2, \ldots, v_n \rangle$, an *attribute vector* $\vec{\alpha}$ is defined as:

$$\vec{\alpha} = \langle \kappa_1 : v_1,\ \kappa_2 : v_2,\ \ldots,\ \kappa_n : v_n \rangle$$

□

Since the keys come out of a set, every key-value pair in the attribute vector can be uniquely addressed. The tuple of values has no such restriction; more than one key can be associated with the same value.

A related operation is called *attribute projection*:

Definition 15

Given an attribute vector $\vec{\alpha} = \langle \kappa_1 : v_1,\ \kappa_2 : v_2,\ \ldots,\ \kappa_n : v_n \rangle$ and κ a key, then it holds that

$$\vec{\alpha} \bullet \kappa = \begin{cases} v_i & \text{if } \exists\, i \mid \kappa = \kappa_i \\ \texttt{nil} & \text{if } \nexists\, i \mid \kappa = \kappa_i \quad \text{(in case } \kappa \notin K_\alpha\text{)} \end{cases}$$

□

The principle of Lexicals is based on a two-stage lookup process. As with Tuples, Lexicals have Class and Instance objects. Lexical Classes @2 contain a fixed set of keys and for each key a procedural description that functions as a default rule. Lexical Instances @3 have a subset of the keys of their Lexical Class and for each key a plain string as value. The lookup mechanism is organized in such a way that each Lexical Instance can overrule a default (Class) rule by having a value with the same key.

Lexical Classes

For every LexCls instance $\in \Omega_2$, there is a function **form** which returns an attribute vector. The associated function $\mathbf{form}(o, \kappa)$ returns the result of attribute projection of κ on $\mathbf{form}(o)$. The values for LexClss are scripts in a practical string manipulation language (Section 3.3.2).

Inheritance is defined in the same spirit as for previous Base Classes; when a key κ is missing in the attribute vector of a TupCls instance, the LMS searches up its ancestor chain until a matching key is found.

Lexical Instances

The same approach as for Lexical Classes is followed with Lexical Instances, except that Instances return character strings out of the associated alphabet as results of the $\mathbf{form}(o, \kappa)$ function.

When, after following the complete ancestor chain up to the top element Δ, the resulting value is still `nil`, the LMS starts processing the script in the associated Lexical Class. If this script is missing, inheritance continues through the Class' ancestor chain.

4.2.8 Feature Overview

The Base Classes such as presented in the previous sections have many features that may be somewhat overwhelming. To help the reader in getting an overview of all Base Class features and to be able to compare the Classes, Table 4.1 contains a matrix of salient features of all Base Classes. I ordered the table rows in approximate order of complexity.

Table 4.1: Base Class Feature Matrix

Base Class	Layer	LA Defined	User Defined	Values	Algorithms	Update Restrictions	Class/Instance
String	2		x	x			
RegExp	2		x		x	x	
Tuple Class	2	x		x		x	x
Tuple Instance	3		x	x		x	x
Set Class	2	x		x			x
Set Instance	3		x	x			x
Lexical Class	2	x			x	x	x
Lexical Instance	3		x	x		x	x

The 'LA Defined' and 'User Defined' columns indicate whether the objects can be created by the Lexicon Administrator (@2) or the Lexicon Users (@3) (they are mutually exclusive). 'Values' and 'Algorithms' indicate whether the objects contain static values or scripts that generate results on request. Objects with 'Update Restrictions' have an update invariant not equal to 'true,' and therefore update attempts must be validated before actual updating.

4.3 Combining LMS Base Classes

In this section, I will present from the point of view of the user how Base Classes can be used to build domain descriptions at the type level.

As explained in Section 4.1.4, a usable Lexicon consists of a mix of instances at layer 2 and 3. Layer 2 provides the used linguistic paradigm. For reasons of clarity, I will use a simplified form of Functional Grammar (Dik, 1989) as the paradigm in this section. Note that this does not imply that the LMS is in any way tuned towards FG; it is meant to be generic. Also at Layer 2, I will present basic English morphosyntax, but without the complete string handling scripts that should be provided to the LMS to handle default morphosyntactic word formation rules; these scripts are described in Section 3.3.2. To have some actual domain of which I can store the terminology at Layer 3, I will use a simple warehouse case. This case is not selected for its complexity, but because it provides typical examples of terminology types, which let me illustrate some basic LMS features while maintaining a clear relationship with known concepts in the real world and in various conceptual modeling paradigms. Where appropriate, I will use NIAM and KISS for more detailed domain models.

4.3.1 Setting up the Lexicon Schema

Before any data can be put in the LMS, a linguist must first set up Layer 2 in such a way that the lexicographers have 'building blocks' to work with. The LMS offers enough flexibility to implement the same linguistic paradigm in various ways. In this section, I present just one way to set up a paradigm layer for (a subset of) Functional Grammar and a language layer for English.

Terms

A substantial part of domain terminology consists of *terms* or common nouns, which largely equate ER entities and KISS/NIAM objects. All these terms represent some domain concepts that are talked about by the domain specialists, both concrete and abstract.

I use an instance of TupCls with a tuple width of 0 to represent terms, following Mackenzie (1987). Terms can be organized in inheritance hierarchies if required. Between the terms themselves there is nothing to inherit (the tuples are empty), but the relationships between terms (expressed in frames, see Section 4.3.1) can also follow this inheritance tree/lattice.

Proper Nouns

In a classic FG approach, there is not much difference between common nouns and proper nouns (except of course for the individual identity of proper nouns). However, in the context of conceptual modeling—and especially for database applications—it is essential to separate them. Whereas common nouns typically are used as column headers, proper nouns turn up as column values. No Lexicon should strive to contain all possible proper nouns that exist in any domain. When a Lexicon would indeed store all individual names of all relevant objects in the domain, it would become the database itself.

The LMS therefore provides two related Base Classes to hold individual identifiers or names: Strings and Regular Expressions. Of these two, Strings are best fitted to contain a sample of names from an open set, such as person names. Regular Expressions should, on the whole, be used whenever there is a closed naming convention, such as with article numbers. In both cases, the extension of the set of names is only meant to give some example values and not to be exhaustive, although the actual extensions (especially with regular expressions) could be covering the domain.

Frames and Roles

Verbs are represented in FG through Frames. The LMS provides the generic Tuple Base Classes to cater for this need.

Frames consist of sub-elements called Roles, which in turn contain selection restrictions on the 'actors' (terms) that typically 'play' these Roles. Setting up a Lexicon Schema to support FG Frames therefore requires two steps: first, the definition of all appropriate Roles, and second, the definition of the ways in which these Roles can be combined in Frames.

A Role is a one-place Tuple that contains a Term as a selection restriction. Because the various types of Role (Agent, Patient, etc.) are usually given by the paradigm, or at least need to be controlled, Roles are formed from Tuple Classes @2 and are given a fixed name and selection restriction (usually Term) that broadly indicates the required type of 'actor.' Because it is not possible to refer to actual Lexicon elements @3 from the Paradigm Layer 2, a linguist cannot dictate for e.g. all Agent roles to be 'played' by a certain type of Term, such as Animate. If such a restriction were required nonetheless, the solution would be to subclass the single Term (Section 4.3.1) into more specific Terms, and to use these specific Terms in the Role Tuples.

Frames are made out of Tuple Classes @2 as well, but this time the selection restriction(s) obviously are Roles. The most generic way to define Frames is to have one single Role 'master pointer' that forms the top of the Role hierarchy, and to create a few Tuple Classes with one, two, three, and four Role elements.[2] A lexicographer @3 can now create a Frame of the correct arity, say three, and fill it with three Roles, say Agent, Patient, and Beneficiary. If (s)he wants, (s)he can also give a typical selection restriction to these Roles by placing a reference to a pre-existing Term object into them.

Note that it depends completely on the actual Term hierarchy in the Lexicon @3 whether or not these selection restrictions on Roles have any meaning. They should be interpreted as *typical* examples of the 'most abstract' Term that still can be restrictively used in this Role—any Term that is in the offspring of the 'most abstract' term would be allowed to 'play' this Role as well.

Lexicals

The Lexicals are founded less in the Paradigm and more in the used language. In English, the main Lexicals would be Nouns, Verbs, and Adjectives. Lexicals are separate objects in the Lexicon; they hold only form information, and no meaning information at all. Lexicals must be connected to Term and Frame objects to be useful, and reversed, Terms and Frames can only be of use if they are connected to a particular language (root) form—a Lexical.

To maintain flexibility, I chose not to include a pre-wired set to contain Lexical references in the other Base Classes. Instead, a linguist should set up his/her own linking mechanism between Terms and Frames on one side and Lexicals on the other. The most straightforward, and at the same time theoretically most sound way to connect to Lexicals is by means of a special Tuple, called *Expr* or something related. At Layer 2, a Tuple Class is created that receives the name Expr and contains two elements: a reference to the Term (or Frame[3]) paradigm element and a reference to the appropriate Lexical Class. Actual instances of Terms, Frames, and Lexicals in the Lexicon (@3) now can be linked by placing references to them in the same Tuple Instance instance.

The advantage of using a separate array of Expr objects to represent relationships between Terms/Frames and Lexicals is that it becomes easy to add additional

[2]More than four Roles in a Frame are not commonly observed, see (Dik, 1989).

[3]It might prove convenient to create a common supertype of both Term and Frame, just to unite the Expr relations.

Lexicals when appropriate. For example, in modeling paradigms such as NIAM, Frames often need a reverse Lexical as well, especially when passifying a verb will not do (see Section 5.2). The LMS implementation takes care of an efficient retrieval of the required Expr objects when necessary.

Sets

Sets are meant to store additional information about other objects in the Lexicon, in a way that might resemble the classic object attributes. For example, when the same Term has more than one associated Lexical (which is the case when there is synonymy involved), there will be multiple Expr objects with that Term in the first position. By placing only one of these Expr objects in the Set 'preferred,' it is possible to indicate the preferred Lexical for this Term.

Other straightforward applications of Sets are to indicate which lexicographer was responsible for the addition of a particular object, in which semantic field (domain) a Term or Frame belongs, or to group words together to fix idiomatic expressions. Sets can be placed in hierarchies to facilitate their management.

As with all LMS Base Classes, Sets provide relatively little functionality of their own; they are nothing but building blocks for an effective and efficient storage mechanism. The actual (lexical) application needs to combine the information that can be retrieved out of the LMS.

4.3.2 Example case: a Warehouse

The purpose of this section is not to give a complete formal overview of the Warehouse case, but to describe in informal terms the domain out of which I will draw some terminology to be used in LMS examples.

> The warehouse is a central repository of articles, where articles come in, are stored for a while, and are sent out again. Incoming articles are delivered by a supplier and received by an employee. Outgoing articles are sent by an employee and received by a customer.
>
> Articles are identified by article numbers, which are the same for each article of the same type. Individual articles are not identified separately, but the stored amount of articles per type is registered. Each employee has an employee name, and suppliers and customers have company names.

Given the Warehouse case, some typical Lexicon objects are presented in Table 4.2. I use capitals to indicate object identifiers, but please note that *all identifiers are meaningless.* It is only for the sake of clarity that I use English words for some identifiers in this example; an actual Lexicon would only use meaningless numbers.

The table is not exhaustive, but contains some examples of most categories of lexical elements. I left out the Sets, which might be used to mark certain objects as having specific attributes. The paradigm used is based on FG, with terms, frames, ID-terms (specializations of terms to capture groups of proper nouns), several roles,

Table 4.2: Typical Lexicon Objects in the Warehouse Case

Identifier	Paradigm	Base Class	Values
WAREHOUSE	Term	TupIns	—
ARTICLE	Term	TupIns	—
EMPLOYEE	Term	TupIns	—
RECEIVE	Frame	TupIns	(AG5654,PT9812)
IDENTIFY_1	Frame	TupIns	(AG3434,PT1253)
IDENTIFY_2	Frame	TupIns	(AG5209,PT9976)
ARTICLE NUMBER	ID-Term	RegExp	999999
EMPLOYEE NAME	ID-Term	String	{Jones,McLean,Smith}
AMOUNT	ID-Term	RegExp	9+
LX3234	Noun	LexIns	'warehouse'
LX4534	Verb	LexIns	'store'
LX8243	Noun	LexIns	'article number'
AG5654	AgentRole	TupIns	(EMPLOYEE)
PT9812	PatientRole	TupIns	(ARTICLE)
AG3434	AgentRole	TupIns	(ARTICLE NUMBER)
PT1253	PatientRole	TupIns	(ARTICLE)
AG5209	AgentRole	TupIns	(EMPLOYEE NAME)
PT9976	PatientRole	TupIns	(EMPLOYEE)
EX1234	Lemma	TupIns	(WAREHOUSE,lx3234)
EX6874	Lemma	TupIns	(STORE,lx4534)
EX4223	Lemma	TupIns	(ARTICLE NUMBER,lx3434)

lemmas (links between terms/frames and lexicals), and two simple English word formation rule collections, noun and verb. The LMS also holds the definition of these paradigm elements, but I left them out of the table to concentrate on the Lexicon contents. Note that the proper nouns themselves are also absent; they can be found in the actual database, built according to the data schema, and do not belong in the Lexicon. The few proper noun examples (in the String and RegExp objects) are just examples; they could be used both in generation of example table populations and to parse NL sentences, if required.

4.4 LIX

A Lexicon Management System, or Lexicon Server, cannot function properly without a flexible, standard interface language. The formal notation used in the previous sections is fine for exact definitions, but impractical for any other purpose. Lexicographers working on a large Lexicon need a better way to communicate with the server, and also some way to write down lexical structures, in a convenient manner,

outside the system. Additionally, various application-like access mechanisms should be available for specific tasks, such as convenient Lexicon browsing. All these applications should use the same communication language with the LMS. This section presents the fundamentals of such a generic Lexicon control language.

4.4.1 The LIX Language

Machine-readable languages preferably are linear ASCII strings. The Lexical Information eXchange (LIX) language is based on linear ASCII, but care has been taken that is it well-fitted both for stand-alone and embedded usage. LIX is mainly a list-oriented language.

Interactive, batch, and embedded usage

Relatively few updates and queries will be done purely interactively, with a human user typing in commands on a console device and reading the LMS response. The bulk of the update work will be done in batch, with a program reading some input files and sending update commands to the LMS. A filled Lexicon will be mostly queried by application programs that need lexical information during their work. And because of the inherent complexity of the LMS storage structures, some form of assistance by a local client interface (part of a *Lexicographer's Workbench*) might prove interesting.

For example, when a lexicographer requests all the Concepts that relate to a particular Lexical, the LMS responds with nothing but a set of Concept identifiers. A helpful interface would automatically use this set to retrieve more information about these Concepts, such as all the associated Lexicals. All these applications require an *embeddable* LMS communication language, that can be easily wrapped in a general-purpose programming language. This can be compared to the solutions that have seen the light to include SQL statements in languages such as C and COBOL: the programs build SQL statements in the form of plain ASCII strings, and the set-oriented SQL results are processed by iterative operations on lists. The Lexicographer's Workbench, or LWB, is an important research and implementation subject in the TREVI project (see Appendix C).

Manipulating lists

Because actual LMS updates and queries are mostly restricted to single objects (where a Set is considered an object), straightforward *list representations* form the bulk of LIX's input and output. Sometimes, these lists can be nested, such as in the case of Lexical Instances where each attribute-value pair can be seen as a list element that consists of two list elements itself. When a wrapper programming language has native list capabilities, it would be advantageous to directly connect the LIX instructions to these native structures, instead of always going through the tedious process of mapping the native structures into and out of ASCII strings.[4] The

[4]Even when the mapping is done by in-line macros or a preprocessor, a process that is invisible for the programmer but decreases performance, having native structures at hand would be advantageous.

design of LIX therefore is a balance between LMS-specific expressiveness and easy embeddability in existing programming and scripting languages.

4.4.2 LIX Examples

Obviously, LIX can only be designed to manipulate the Base Classes and instances of these Classes—the various paradigm-specific additions to the LMS at layer 2 are LA-defined. But most updates and queries will need to be expressed in layer 2 elements. LIX provides for a transparent translation between layer 2 elements and the Base Class elements @1.

LIX has two sub-languages, which serve two target communities. The first sub-language is called the Lexicon Definition Language (LDL), used by linguists and LAs to define layer 2. The second sub-language is the Lexicon Query Language (LQL). Both sub-languages are combined in the specification and can be used in an interleaved fashion by any application, as long as the user has sufficient rights on the LMS to change layer 2 elements. Naturally, some changes @2 have so much effect on the Lexicon contents @3 that they would either be impractical or lead to loss of information, but these considerations are more an implementation issue and will not be discussed here. The same holds for the technically essential, but conceptually uninteresting maintenance language, that can be used to keep the LMS healthy and to gather statistics on the database.

The LIX LDL

The LDL is meant to create new LMS objects at Layer 2. The following block shows a shortened LIX fragment that will define the FG-like paradigm and two English lexicals, and sets up the necessary linking structures.

```
# Abstract classes, used to give all lexicals and terms/frames
# common ancestors.
lexical word (parent=none forms=root)
tuple concept (parent=none)

# Term and noun.
tuple term (parent=concept)
lexical noun (parent=word forms=singular,plural)
script noun singular
  SCRIPT
  root
  SCRIPT
script noun plural
  SCRIPT
  root+"s"
  SCRIPT

# Some roles.
tuple role (parent=none)
tuple agent (parent=role elements=term)
```

```
tuple patient (parent=role elements=term)
tuple recipient (parent=role elements=term)

# Frame and verb.
tuple frame0 (parent=concept)
tuple frame1 (parent=concept elements=role)
tuple frame2 (parent=concept elements=role,role)
tuple frame3 (parent=concept elements=role,role,role)
tuple frame4 (parent=concept elements=role,role,role,role)
lexical verb (parent=word forms=1sp,2sp,3sp,1pp,2pp,3pp)
script verb 1sp
  SCRIPT
  root
  SCRIPT
script verb 2sp
  SCRIPT
  root
  SCRIPT

# Expressions/lemmas.
tuple lemma (parent=none elements=concept,word)
```

To query and remove definitions, LIX provides some more statements. Most definition changes will lead to considerable information loss in the Lexicon, so measures should be taken to save a current Lexicon and re-load the contents, possibly after an external conversion. There is one exception: the scripts inside the Lexicals can be added, changed, and removed at will, because they are only used for transient processing in case the stored Lexicals @3 fail to provide stipulated word forms.

In the example, I will prefix the returned strings with >>.

```
definition
>> word,concept,term,noun,role,agent,patient,recipient, \
   frame0,frame1,frame2,frame3,frame4,lemma
definition noun
>> lexical noun (parent=word forms=singular,plural)
definition noun singular
>> root
```

I refer the reader to the LIX grammar specification (Section 4.4.3) for more information about these statements.

All definitions need to include a (unique) reference name that will be used for all further references to this particular definition, both within the LDL and the LQL. Specifically, there are no object identifiers involved at all (except, of course, inside the LMS itself).

The LIX LQL

The LQL is meant to create new LMS objects at Layer 3 and to query the resulting lexical database. The next example creates a partial representation of the Warehouse

Case. Because it would be extremely impractical to give each created LMS object a name, each new object returns its object identifier, which must be retained by the application using LQL for future references. Usually, this will not cost much memory, because most object identifiers can be 'forgotten' as soon as a cluster of objects is created and linked together. After the cluster creation, only the lexical form and the top elements of any defined tree remain as handles.

In the example, I will prefix the returned object identifiers with >>. Note that in a real application, the identifiers would be numeric; for the sake of clarity, I will use some mnemonics (in capitals) instead.

```
insert noun (parent=none forms=(root=warehouse))
>> NOUN_WAREHOUSE
insert term (parent=none)
>> TERM_WAREHOUSE
insert lemma (elements=TERM_WAREHOUSE,NOUN_WAREHOUSE)
>> LEMMA_WAREHOUSE

insert noun (parent=none forms=(root=article))
>> NOUN_ARTICLE
insert term (parent=none)
>> TERM_ARTICLE
insert lemma (elements=TERM_ARTICLE,NOUN_ARTICLE)
>> LEMMA_ARTICLE

insert verb (parent=none forms=(root=store))
>> VERB_STORE
insert agent (parent=none elements=TERM_WAREHOUSE)
>> AGENT_WAREHOUSE
insert patient (parent=none elements=TERM_ARTICLE)
>> PATIENT_ARTICLE
insert frame2 (parent=none elements=AGENT_WAREHOUSE,PATIENT_ARTICLE)
>> FRAME_STORE
insert lemma (elements=FRAME_STORE,VERB_STORE)
>> LEMMA_STORE

insert name (parent=none)
>> NAME_FAMILY-NAME
addto NAME_FAMILY-NAME Jones,McLean,Smith
>>
```

String and set additions, such as in the last lines of the example, do not return object identifiers, because the added string/set elements have no identity—they are just members of a set and can only be retrieved as part of the complete extension of the set.

The LMS will respond with (lists of) object identifiers to most queries. Currently there are no provisions to combine several LIX LQL statements into one, so that the application should issue several statements and do some processing to get to

the answer. In future versions of LIX and the LMS, a more complex query language could be provided that can concentrate most processing on the lexicon server and decrease the network traffic.

First some examples of straightforward object queries.

```
query oid
>> NOUN_WAREHOUSE,TERM_WAREHOUSE,LEMMA_WAREHOUSE,NOUN_ARTICLE, \
   TERM_ARTICLE,LEMMA_ARTICLE,VERB_STORE (...)

query oid LEMMA_ARTICLE
>> lemma (elements=TERM_ARTICLE,NOUN_ARTICLE)
query oid FRAME_STORE
>> frame2 (parent=none elements=AGENT_WAREHOUSE,PATIENT_ARTICLE)
query oid NOUN_WAREHOUSE
>> noun (parent=none forms=(root=warehouse))
query oid NAME_FAMILY-NAME
>> name (parent=none extension=Jones,McLean,Smith)
```

It is often necessary to request a complete list of the instances of particular Layer 2 classes. These queries are part of the LQL and not of the LDL, because they do not operate on the definitions but on the *instances* of the definitions.

```
query instances word
>> NOUN_WAREHOUSE,NOUN_ARTICLE,VERB_STORE
query instances lemma
>> LEMMA_WAREHOUSE,LEMMA_ARTICLE,LEMMA_STORE
```

Tuples are often used as look-up tables, and it would be very cumbersome to request the complete extension of all instances of a particular Tuple type and then scan through these instances one by one. Therefore the LMS offers an optimized look-up mechanism for Tuples, which considers the extension of all Tuples of the same type to be a standard tabular relation. The column is referred to by name (as given in the definition) or by index number (where the first column has index 1) when there are more columns with the same name, such as with Frames.

```
query lookup lemma 2 NOUN_WAREHOUSE
>> LEMMA_WAREHOUSE

# In case of synonymity:
query lookup lemma word NOUN_BANK
>> LEMMA_MONEYBANK,LEMMA_FURNITUREBANK,LEMMA_RIVERBANK
```

Lastly, an implementation of the Lexical class must include a specialized look-up table which is heavily tuned towards efficient retrieval of word forms. All possible word forms (both stipulated and generated through scripts) are merged into one single index,[5] and a query to this index returns all matching lexical forms.

[5] An actual implementation might take various shortcuts to save on memory, as long as the result functions like a single unified word form list.

```
query form warehouses
>> NOUN_WAREHOUSE:plural

# Extra example, showing synonymy at the word form level:
query form rain
>> NOUN_RAIN:singular,VERB_RAIN:1sp,VERB_RAIN:2sp (...)
```

4.4.3 EBNF grammar of LIX

A complete Extended Backus-Naur Form grammar will be presented below. This grammar is more extensive than the feature structures in the former section suggest, because LIX is an implementable language, meant for actual communication with and control of a working LMS.

LIX assumes that the usual spaces, tabs, and newlines are all white space and can be ignored except inside quotes, where they will be regarded as single spaces. A LIX parser would need to recognize the tokens which are written between double quotes in the grammar (for example, "lexicon") and treat instances of the `STRING` primitives as tokens with an additional value.

The EBNF grammar could easily be extended to an attribute grammar, with explicit parameter passing between terminating elements and non-terminating elements.

The Lexical Scanner

The lexical scanner divides the input stream into known tokens fitted for the LIX parser. It also handles white space, comments, and other low-level features. Whitespace is formed by spaces, tabs, or newlines. Comments are started by '#' and run until the first newline.

```
'='  = <equals>                  'regexp'    = <regexp>
''' = <quote>                   'string'    = <string>
'('  = <paren0>                  'set'       = <set>
')'  = <parenC>                  'tuple'     = <tuple>
','  = <comma>                   'lexical'   = <lexical>

'script'     = <script>          'query'     = <query>
'definition' = <definition>      'oid'       = <oid>
'undefine'   = <undefine>        'instances' = <instances>
'insert'     = <insert>          'lookup'    = <lookup>
'addto'      = <addto>           'form'      = <form>
'delete'     = <delete>
'removefrom' = <removefrom>

<char>    --> ('a'..'z' | 'A'..'Z' | '0'..'9' | '_' | '-' | '.')
<digit>   --> '0' .. '9'
<charstr> --> <char> {<char>}
<number>  --> <digit> {<digit>}
<qstr>    --> <quote> <charstr> {<charstr>} <quote>
```

```
<str>     --> <charstr> | <qstr>
<avp>     --> <str> <equals> <str> {<comma> <str>}
<navp>    --> <str> <equals> ( <str> {<comma> <str>} |
                               <paren0 { <navp> } <parenC> )
```

The tokens in the lower group (<char> to <navp>) have an associated value that is passed on by the lexical scanner. Strings may be quoted or not; quotes protect white space.[6] Attribute-value pairs contain one key string and one value, which may be a list. Nested AV-pairs also have one top level key, but may have sub level keys and nest indefinitely deep.

The <char> set of characters could be extended with e.g. diacritical characters such as ë, ñ, and ø, or any other character that would be useful to represent lexicals. It would not be a true problem for the LMS to store non-Latin characters, such as Greek, Japanese Hiragana and Katakana, or even Kanji, but the usual character-oriented computer platforms might pose difficulties here. UNICODE promises some universal character encoding at the expense of double memory requirements.

The LDL Syntax

For clarity, I will break up the complete production rule for <ldl> into smaller chunks. The BLOCK nonterminal is a special construction to guard embedded scripts in any convenient language. Essentially it is divided into three parts: a random ASCII string followed by a newline, the script, and a repetition of the ASCII string and the new line. The string can be chosen by the LDL author so that it does never interfere with the script language; in between the sentinel strings, each character is picked up by the lexical scanner without any post-processing.

```
<ldl>      --> ( <regexp> | <string> | <set> |
                 <tuple> | <lexical> ) <str> [ <features> ]
<ldl>      --> <script> <str> <str> BLOCK
<ldl>      --> <definition> [ <str> [ <str> ] ]
<ldl>      --> <undefine> <str> [ <str> ]

<features> --> <paren0> { <avp> } <parenC>
```

The LQL Syntax

```
<lql>      --> <insert> <str> [ <nestfs> ]
<lql>      --> <addto> <str> <str> { <comma> <str> }
<lql>      --> <delete> <str>
<lql>      --> <removefrom> <str> <str> { <comma> <str> }
<lql>      --> <query> <oid> <str>
<lql>      --> <query> <instances> <str>
<lql>      --> <query> <lookup> <str> ( <number> | <str> ) <str>
<lql>      --> <query> <form> <str>

<nestfs>   --> <paren0> { <navp> }  <parenC>
```

[6] The parser replaces white space of any form with a single space.

4.4.4 LIX Implementation

Up to the date of this writing, LIX has been partially implemented in the TREVI Project (see Appendix C). The LDL part of LIX has been implemented with very few changes, most notably the absence of regular expressions and strings (because TREVI does not need any name examples). The LQL part of LIX is not implemented as an ASCII language but as a series of Java methods that follow the same basic structure. Appendix C presents an LDL specification of the language-independent semantics, complete morphosyntactics for English, and partial morphosyntactics for Spanish.[7]

[7]Spanish has been completely described in TREVI, but the specification is too lengthy to include in this thesis.

Part II

Applications

Chapter 5

The Lexicon and Object Role Modeling

— in which the writer presents NIAM *and* NORM *in the context of linguistic analysis. He discusses some heuristics and formal approaches to support both models with a Lexicon and suggests a fundamentally different way to model a domain, using linguistic structures as the base instead of logical (relational) structures —*

This chapter gives an overview of the basic concepts of NIAM, one of the leading ORM modeling languages and information analysis methods, and its object-oriented cousin NORM. I present a brief semi-formal basis of both modeling languages, partially inspired by Conceptual Graph Theory, followed by various subjects related to the usage of natural language in NIAM and NORM and the possibilities for mapping their diagrams to natural language and vice versa.

5.1 NIAM as an analysis tool

I base my description of the NIAM formal specification language on the publications of Nijssen and Halpin (1989), Halpin (1995), and De Troyer (1993). Halpin's new variant of NIAM, called FORM, is not significantly different from the 1989 version, and I will take the liberty to call them both 'NIAM.' Nijssen currently presents NIAM as being more than 'just a way of drawing pictures' and calls the approach 'Universele Informatiekunde' (Dutch for Universal Information Science, which is too ambitious a name in my opinion). In his UI book (Nijssen, 1993) he also includes explicit procedures to elicit NIAM structures from users and to transform them into information system designs. UI does not significantly enhance traditional NIAM as far as notation or underlying concepts are concerned, and therefore I will stick to the NIAM and object-role modeling publications in English. De Troyer's own NORM, an OO variant of NIAM, will be discussed in Section 5.3.

5.1.1 An Overview of NIAM

In the early seventies, and based on the ideas of Fillmore (1968), Falkenberg proposed to base conceptual schema concepts on elementary natural language sentences. Nijssen then introduced the approach of building a database design by starting with

specific examples (Nijssen and Halpin, 1989, p.31). A first description of a 'Binary Relationship Model' was given in 1974 by Abriel, at a time when Entity-Relationship modeling (Chen, 1976) was still in its infancy. For several years, the binary relationship model was developed independent of ER. The name 'NIAM' first surfaced in 1976 (Senko, 1976) as an acronym for *Nijssen's Information Analysis Methodology* (Verheijen and van Bekkum, 1982; Nijssen and Halpin, 1989, p.31).

The model was developed further by Nijssen, Falkenberg, Meersman, Halpin, and ter Hofstede, among others (Nijssen and Halpin, 1989). Currently, NIAM-based formalisms are known under several names, with slightly different notations but comparable semantics. Some extensions have been proposed, notably nesting (nominalization of facts) and n-arity. Some approaches stress acquisition and presentation, others theoretical fundamentals and automatic SQL table definition generation. The Predicator Set Model (PSM) variant currently is the most generic one (ter Hofstede et al., 1993).

NIAM and NL

When introducing UI, Nijssen re-christened the method to *Natural-language Information Analysis Methodology* (Nijssen, 1993), but recent publications (Kim and March, 1995) still present the original variant of the name. Because of Fillmore's ideas, and because examples are presented in NL, NIAM indeed has some NL influences, but to call it 'Natural-language Information Analysis Method' in my opinion is not appropriate. 'Example-oriented' would be a better description.

Still, compared to other conceptual modeling approaches, NIAM has a relatively strong background in natural language. This is mainly because, in its early development phases, it was used to model existing table structures to support reverse engineering. The *telephone heuristic* (Nijssen and Halpin, 1989, p.33) to paraphrase tables as if reading them aloud to someone else through a telephone was invented for this. Later on, NIAM became a development method for new tables, but it retained some of the NL approach. NIAM diagrams still are mostly paraphrasable in NL, although there are several fundamentally different methods to do this, depending on the actual words chosen in the NIAM diagram (see section 5.2).

Application and Support

Due to its lack of any dynamic description tools, NIAM still is mainly a *data* modeling method, used to specify the static part of a conceptual model. The nature of its approach does away with the need for careful normalization of the resulting data models (Nijssen and Halpin, 1989, p.273), and it can be properly mapped to a logical table model with automated procedures. Several tools exist to translate NIAM models into data definition languages for commercially available (relational) database systems, see De Troyer et al. (1983), Asymetrix (1994), and Halpin (1995).

Some specification and data manipulation languages have been developed which use a NIAM-based formalism. RIDL (Reference and IDea Language) was developed in the 1980s at Control Data Belgium, and has been implemented in a commercial CASE tool called RIDL* (Meersman, 1982). LISA-D (Language for Information Structure and Access Descriptions) is a more generic version of RIDL, bound to the generic

ORM formalism PSM, developed by ter Hofstede et al. (ter Hofstede et al., 1993). FORML, implemented in InfoModeler (Asymetrix, 1994), is a comparable specification language. For a more complete overview of ORM specification languages and tools, see (Halpin, 1995).

5.1.2 NIAM formalized

The following description is partially based on (de Troyer, 1993, p.90). She describes a Binary (object-)Role Modeling approach (BRM) from a theoretical point of view, ultimately leading to strict formal transformation rules capable of generating SQL tables. The RIDL CASE tool (de Troyer et al., 1983) was largely based on this work.

Some variants of NIAM use not only binary roles, but also n-ary roles and/or nesting. I will discuss these techniques in Section 5.2.5.

In BRM, all the data is modeled as *object types* and *binary relationships* between object types. Binary relationships are also called *facts*. Typical for the NIAM BRM is its explicit distinction between lexical and non-lexical object types. Lexical object types (LOTs) have an equivalent representation in a language, while NOLOTs have not. A typical LOT is *person-name* (both the intension and the extension of the set of all person names), a typical NOLOT is *person*. Object types can be ordered in a hierarchy, by means of a special binary relationship called a *sublink*. A BRM schema is a data schema for which the following conditions hold:

1. Types are divided in two disjoint groups: LOTs and NOLOTs.
2. Only NOLOTs can be part of the object hierarchy.
3. All relations are binary (i.e., have exactly two roles).
4. Relations can be divided into two disjoint groups: facts and sublinks.
   (a) All facts involve at least one NOLOT.
   (b) A sublink relation exists between each two types for which the subtype relation holds.
   (c) Each LOT may be involved in at most one fact relation.
5. All instances in a role of a relation are in the extension of the type of this role.
6. In a populated BRM schema there can be no null values, and for each instance it can be decided whether it is an instance of the type or not.
7. Sublinks are special binary relations:
   (a) If an instance of a supertype extension is also an instance of the subtype extension, then it should occur as a tuple in the sublink relation.
   (b) A sublink relation can only contain tuples with equal first and second elements.

A *path* between two views is a sequence of one or more binary relations where each two successive relations are connected through the same type or through types of the same family. Paths can be viewed as relations, and can be reversed (*inverted*). In addition, it is possible to isolate the first relation in a sequence, the rest of the sequence, and the last relation in the sequence. A path is called *proper* if the last element of it is a fact.

NIAM has several declarative constraint types. For a complete formal treatment of these, I refer to (de Troyer, 1993).

The Uniqueness Constraint states that an instance of an object type can be identified by a combination of object type instances specified through a number of paths.

The Identifier Constraint is a special case of the Uniqueness Constraint, where only one path or fact is involved.

The Total Union Constraint specifies that the existence of an object type instance depends on it being related to at least one of a number of other object type instances, specified via a given path, or on its existence as an instance of some subtype.

The Total Role Constraint is a special case of the Total Union Constraint, where only one path or relation is involved.

Path Subset, Path Equality, and Path Exclusion Constraints express the inclusion, equality, and exclusion of the extension of two paths.

Role Subset, Role Equality, and Role Exclusion Constraints specify that instances participating in a path must also participate in another path, that instances participating in one path must also participate in the other path and vice versa, and specify the mutual exclusion of instances in different paths, respectively. The Role Exclusion Constraint is also applicable on sublinks.

A BRM schema needs to be *complete*, meaning that types with a common subtype must have a common supertype, and *consistent*, meaning that there must exist at least one valid database instance of it. In addition, usually it is required that a BRM schema is *lexically referenceable*, meaning that it must be possible to refer to each NOLOT by means of one of more LOTs.[1]

5.2 NIAM and Frames

In this section, I look at NIAM from the viewpoint of frame representations. Because Conceptual Graphs (CGs) are a convenient way to present frames (Section 3.4.3), I

[1]This requirement has its origin in the fact that the BRM is traditionally used as a conceptual data model for a *relational* database. Because of technical reasons, all NOLOTs have to be replaced by LOTs in such an implementation. Object-oriented implementations have no direct need for such a requirement.

will use CGs to convey most frame notions. I will mostly use frames in the same way as Functional Grammar handles predicate frames (Section 3.2.1), i.e., in their most abstract appearance, at type level.

In Section 5.4, I will discuss NIAM and NORM from a linguistic perspective, using language notions to extract NIAM structures from natural language. The current section concentrates on linguistic features available in traditional NIAM, and emphasizes the various problems that we encounter when trying to formalize NIAM's language connection.

5.2.1 Sentence Conventions

The base primitives of NIAM are called NOLOT, LOT, *fact*, and *sublink*. Paraphrasing of NIAM structures commonly involves using a noun for LOTs and (the intention of) NOLOTs and a 'connector' word in between, such as in 'employee has gender' and 'employee works_for department.'

Although it may appear to the casual observer that the connector word is always a verb, which is also suggested by common ER heuristics (Chen, 1983), this is not the case. NIAM does not specifically require a verb in this place, as exemplified by 'person has smoking_status' versus 'smoking_status of person' (Nijssen and Halpin, 1989, p.52). Some publications have a clear preference for prepositions as connectors or use almost complete phrases: 'department which_does_employ employee', (in Dutch: 'afdeling met_als_medewerker werknemer'), 'rubriek secundaire_rubriek_van boek', 'department with_a_budget_of amount_of_money' (Wintraecken, 1985). Moulin and Creasy (1992, p.173) say about these aspects:

> "The names of the roles within fact types or reference types are chosen intuitively, although the practice is to pick the name of the verb in the sentences used to describe elementary propositions. This practice can sometimes lead to associations between roles which impair the cleanness of the formalism. [...] Hence in their current form, NIAM data schemas cannot be directly mapped to semantic structures which could be used by natural language processors."

NIAM always requires sentences which represent a *predicate* of some arity (usually two or higher), because ORM is fundamentally based on (first order) predicate logic (Gamut, 1991). All of the previous examples convey a predicate, and languages such as FORML are built around n-ary mixfix predicates (Halpin, 1995, p.50). However, the way in which the examples paraphrase the predicate core differs greatly, as already observed by Moulin and Creasy. Although all NIAM facts *could* be expressed as a verb phrase with a subsequent string of object-role combinations, an approach advocated by Halpin, not all NIAM elicitation methods see this as a strict requirement.

I can conclude that NIAM, although using natural language to a certain extent, does not give clear and consistent rules about *what* language should be used. As long as facts can be specified with it, any sentence structure and combination of word types is allowed. This flexibility makes paraphrasing NIAM suitable to a number of languages and applications, as exemplified by the next example in both English

and Japanese (Halpin, 1995, p.51), where italics indicate LOTs and boldface the predicates:

> The Employee with *employee#* '37' **works in** the Department with *name* 'Sales'.
>
> Jogyo in *jugyo in bango* '37' **wa** 'Eigyo' to iu *namae* no Ka **ni shozoku suru**.

Note that the basic elements are the same, but that their placement and distribution varies between languages.

A more linguistic approach prefers a rigid structure to represent the predicate, a structure that is completely language-independent and related to semantics only. Linguistic frameworks such as FG use the frame for this, as do conceptual languages such as RIDL. The advantage of these language-independent semantic structures is that they can easily be compared and related to each other, because the relevant predicates and terms are centralized in a Lexicon. Actual paraphrases can be generated out of the frame representations by expression rules, so the analysts are not always required to work with unnatural (logic) expressions. *Ad hoc* selection of suitable sentences might seem more convenient, but in the long run the collection of frames and concepts in the Lexicon in my opinion will provide a better base to build models on. Observe that I do not claim that NIAM's fundamentals (predicate logic) are not as rigid as linguistic frames; I merely claim that the less restricted paraphrasing rules of current NIAM-based formalisms offer less opportunities for exploitation of terminology semantics.

5.2.2 Generic NIAM Frames

The particularities as sketched in the former paragraph lead to a problem when trying to capture NIAM in a generic frame structure, depicted by a conceptual graph. An intuitive approach would be to see any NIAM sentence as a frame with one predicate, two roles, and two terms, presented in conceptual graph (CG) notation as:

$$[\text{OBJECT}] \leftarrow (role) \leftarrow [\text{CONNECTOR}] \rightarrow (role) \rightarrow [\text{OBJECT}] \qquad (5.1)$$

But if the NIAM sentence does not contain a verb as the connector word, then I cannot find proper CG roles (note the difference between CG roles such as AGENT and MATERIAL, which are *between words*, and NIAM roles, which are represented by the connector word and, in fact, are a practical paraphrasing of a predicate). A traditional, well-formed NIAM fact such as 'department with employees' has no simple equivalent in standard conceptual graphs, because the sentence does not contain a verb. I question the solution to include empty or zero roles, as I think that this severely undermines the semantic content of the representation. Moulin and Creasy (1992) use empty (NULL or EQVL) roles, but they offer the option of refining the model by replacing the NULL relations with (extended) Conceptual Graph case grammar relations. A drawback of this approach is the fact that such a replacement action involves extra semantic analysis in order to find the appropriate semantic roles, which are not in any way implied by NIAM.

A more meaningful approach always uses verbs for the connectors. This makes (CG) role assignment both possible and useful. I now have three possibilities to convert NIAM structures to conceptual graphs:

1. Link two (CG) objects directly with one arbitrary (CG) role.
2. Use two fixed standard roles and an arbitrary verb in between.
3. Use two arbitrary roles and an arbitrary verb in between.

A fourth possibility, two arbitrary roles with a fixed verb in between (always paraphrased as 'object X relates_to object Y') is ruled out because this would mean both throwing away most of the semantics of a relation and introducing a fully superfluous verb.

One arbitrary role, no connector

The first approach would transform CG 5.1 into

$$[\text{OBJECT}] \leftarrow (\textit{role}) \rightarrow [\text{OBJECT}] \tag{5.2}$$

Such a representation has the disadvantage of lacking any connector word (usually a verb), since the typical (CG) roles such as AGENT have no direct equivalent utterance. I would sever the connection between NIAM NL sentences and conceptual graph pronunciation by following this route. Moulin and Creasy have taken the reverse approach and give some (E)CG conceptual relations the same name as the NIAM fact type, especially for simple object attributes. This introduces a large number of arbitrary[2] 'semantic roles.' The CG formalism allows for an unlimited number of roles; they are not an integral part of the theory (Section 3.4.3). However, such an approach ignores the obvious parallels with natural language—it blocks consistent paraphraseability. Furthermore, the decision to represent some relations with a triple $\langle \text{role}_1, \text{verb}, \text{role}_2 \rangle$ and others by a single role leads to an inconsistent paradigm. In my opinion, using only a single, but specialized role has to be viewed as a notational shortcut. It is a syntactical variant, convenient as long as the analyst is aware that (s)he in fact needs a (possibly standard) triple. Introduction of a new 'shortcut' or 'abbreviation' symbol with a legend next to the diagram would serve the same purpose without causing inconsistency.

We here run into the classical problem that not every possible predicate has an associated word, e.g., IS_MOTHER_OF needs to be paraphrased with the help of a copula. See the Sections 5.2.3 and 5.2.4 for more discussion of this subject.

Two standard roles, one arbitrary verb

The second way to encode NIAM in frames is to find proper (CG) roles for each type of standard NIAM structure and stick to these roles for any combination of nouns and verbs. The only two combinations that are of interest are a NIAM fact with two NOLOTs and a NIAM fact with one NOLOT and one LOT. Although NIAM

[2] Arbitrary in the sense of 'invented on the spot by the analyst.'

has the concept of a combined LOT/NOLOT, this combination often leads to an increase of the semantics of the connector in an effort to capture the information lost by the combination. An example of such an enriched connector is 'people work_for_departments_identified_by department_names' (where the LOT has been chosen as the most important concept of the LOT/NOLOT combination). Unfortunately, these particular connector semantics are heavily dependent on the way in which the analyst decides to capture the lost semantics in arbitrary words, and cannot easily be standardized. But since the LOT/NOLOT case is only a notational convenience that can be split up in two normal facts, I will concentrate on the NOLOT–NOLOT and NOLOT–LOT fact types, both of which can be treated the same. The basic structure, then, equals that of conceptual graph 5.1, with the objects replaced by nouns and the connector by a verb, and standard roles for all graphs:

$$[\text{NOUN}] \leftarrow (std\_role_1) \leftarrow [\text{VERB}] \rightarrow (std\_role_2) \rightarrow [\text{NOUN}] \tag{5.3}$$

The reason that standard role assignment, valid for every NIAM expression, is tricky at best, is that NIAM traditionally does not assign *any* semantics to the words it uses. The meaning of a fact is irrelevant, only its *existence* is. Moulin and Creasy (1992, p.173) say about this aspect of NIAM conceptual modeling:

> "The semantics of the fact types are expressed informally by their naming and informal definitions in the data dictionary. Formal languages are not used to give any meaning to the fact types."

Furthermore, NIAM facts are reversible. For binary fact types, fact reversal usually involves a change of the verb from active to passive voice, an exchange of the object and subject roles, and some syntactic sugar: 'employer employs employee' becomes 'employee is_employed_by employer'. But often, fact reversal has to exchange both the agent/patient and the subject/object roles at the same time, and change the associated verb as well: 'employee works_for employer' becomes 'employer employs employee', and not '$*$ employer is_being_worked_for_by employee'. And when the NIAM role is only represented by a single preposition or a phrase, as in 'employee was born in year', fact reversal cannot be done without complete semantic understanding.

As an aside, polyadic NIAM fact types with a predicate and more than two roles cannot easily be reversed, because there is no inherent ordering in the roles—the roles each have their own role English usually a preposition. FG frames are restricted to a small set of essential roles,[3] while more roles can be attached at will as satellites without fundamentally changing the predicate's meaning. NIAM does not acknowledge this linguistic difference between basic roles and satellites.

When the linguistic 'main roles,' typically AGENT versus PATIENT, would be known in a fact type, fact reversal would become easier and the other roles in the predicate would be clearly set apart as satellites. NIAM's adherence to path navigation from any role (important for conceptual query by navigation) prohibits such a special position for the semantically most important roles, and thus decreases the possible knowledge content of a fact type.

[3] Essential in the sense that they determine the meaning of the predicate; see the 'to bar' example on page 24.

Two arbitrary roles, one arbitrary verb

The third and last way to encode NIAM in frames would be to leave the choice of both the roles and the verb up to the analyst. Although this gives the analyst the most freedom, it would also triple the analytical effort that the analyst has to put into the connections, and the advantages would be questionable. However, this approach would become feasible if the analyst could pick a verb and an automated system, i.e., the Lexicon, would supply the roles (possibly by presenting a pick list of semantically allowed possibilities in case of polysemy, see Section 3.2.1).

5.2.3 More on Connector Words and Phrases

Often it is attractive to create sentences that do not fully follow the basic sentence structure NOUN–VERB–NOUN. For example, the reverse form of 'Assemblies consist of elementary parts' might intuitively seem to be 'Elementary parts are parts of assemblies'. Linguistically, this is correct, but there is a problem with the expression '... are parts of...'. This is not a simple or complex verb, but a verb *phrase*, where the verb itself (a form of BE) is not at all the main meaning carrier.

The verb BE is not often used as a main verb, describing a relationship between concepts. If so, the relationship usually is one of subtyping: 'Employees are people', and this is only a less precise way of writing 'Employees are a kind of people', indicating that employees have some place in the class of all people (Quirk and Greenbaum, 1973). But in most cases, BE by itself does not carry meaning. The example above shows that the true meaning of the fact is contained in the construction 'part(s) of', and that 'are' is just a 'linker.' The term *copula* is generally used for verbs used in this fashion, see Quirk and Greenbaum. It seems inappropriate to honor BE as the main carrier of meaning in this case, and thus to equate the NIAM fact with a frame based on BE.

There are other copula verbs, such as APPEAR, FEEL, LOOK, REMAIN, and SEEN, which all carry additional semantics with a personal bias. Preferably, these verbs are never used in the description of facts. Other copulas are related to people's senses or to temporal aspects, and are less fit for use in elementary sentences in general (Quirk and Greenbaum, 1973).

Sometimes a careful selection of another verb may solve the representation problem, e.g., using the verb FORM without any proposition conveys the same meaning (but reversed) as BE_PART_OF in the previous example: 'Elementary parts form assemblies'. But there remain some cases in which a single (complex) verb does not work, while the intended meaning of the fact type (the predicate) is clear and should be supported. In the next section I will present some general aspects of three major verbs that need special treatment, and thereafter I will present some other ways to generically paraphrase fact types which do not consist of one single verb.

The primary verbs BE, HAVE, and DO

Both BE, HAVE, and DO are unusual verbs. In many languages, words comparable to these three are the cornerstone of more complex constructions, and as such have lost most of their original meaning. For example, in English these primary verbs used as auxiliaries share an association with the basic grammatical verb categories

of tense, aspect, and voice (Quirk et al., 1985, p.129). I will give a short overview of the main features of the three primary verbs, also based on Quirk et al.

The verb BE is a main verb (with a copular function) in sentences such as 'Ann *is* a happy girl' and '*Is* that building a hotel?' These sentences convey the traditional IS_A semantics, where it is not yet clear which of the three forms of IS_A (analytical generalization, role playing, or class/instance relationship) is intended. But BE also has two auxiliary functions: aspect auxiliary ('Ann *is* learning Spanish', 'The weather has *been* improving') and passive auxiliary ('Ann *was* rewarded a prize', 'Our team has never *been* beaten').

HAVE also functions both as an auxiliary and as a main verb. As an auxiliary for perfective aspect, HAVE may be used to form complex verb phrases: 'I *have* finished', 'What *has* she bought?', 'They may *have* been eaten'. As a main verb, it normally takes a direct object, and has various meanings such as possession: 'I have no money'; 'They had two children' etc. Often it is encountered in combination with DO, as in 'We *don't have* any money', and with *got* as in 'John *has got* courage'.

As a main verb, DO can combine with a pronoun object to act as a pro-predication referring to some unspecified action(s). The pronoun object may be personal (*it*), demonstrative (*this/that*), interrogative (*what*), or indefinite (*nothing/anything*, etc). Apart from these uses as a pro-form, the main verb DO has a wide range of uses as a general-purpose agentive transitive verb, especially in informal speech: 'She's *done* some really good essays'. DO in such sentences is often replaceable by a verb of more exact meaning; e.g. in SERVICE or MAINTAIN, and in WRITE.

While BE and DO have a rather straightforward field of application, HAVE is much more troublesome. The ownership relation between two concepts connected by HAVE (as in 'employees have employee numbers') is much more generic as with most other verbs, giving HAVE some kind of *one-size-fits-all* status. Weigand (1990) gives an illustration of this flexibility in the context of the so-called 'belonging fallacy':

> "The 'belonging fallacy' is the idea that a given element should be awarded a particular slot in our representation because, in English, we would be given to say that the slot 'belongs to' that element. Thus persons *have* names, ages, and addresses; physical objects *have* weight and height; and so on. Wilensky (1986) calls this a fallacy because the underlying concepts of 'belonging' have almost no context-free meaning.
>
> For example, consider the phrases 'John's apartment', 'John's car', and 'John's girlfriend'. While formally similar, these phrases have radically different interpretations. [...] The conclusion is that frame/slot representations assert that there is a relationship between two entities, but the relationship is arbitrary and has no representational status."
>
> (Weigand, 1990, p.80)

In many cases, HAVE does not at all indicate any ownership or part-of relationship, but a kind of generic relationship, carrying *is-associated-with* semantics without specification of the kind of association. In sentences written by domain experts, one will often encounter cases of HAVE where other verbs would be more

appropriate. An example would be: 'elementary parts have part numbers'. In everyday life, this generic use of HAVE poses no immediate problems, since our semantic knowledge tells us that elementary parts have no true ownership relation to part numbers, hence HAVE must have been used as a generic relation describer. But in this particular case, the reverse sentence is a lot more substantial: 'part numbers identify elementary parts' explicitly tells us about the *identify* relationship.

When true meronymy (part-of) relationships are intended, various possibilities exist to correctly paraphrase the intended meaning without the use of HAVE (Storey, 1991), such as 'component of object', 'member of collection', and 'portion of mass'—see also Figure 5.1. Unfortunately, the verb in all these cases is BE, used as a copula, and therefore it carries no meaning. The next sentence discusses such verbless sentences.

5.2.4 Verbless Sentences and Predicate Formation

After all the previous examples of sentences with verbs as the connector word, with various levels of complexity, there still remain some cases in which the NIAM fact cannot properly be described with a simple verb. For example, there is a problem with relations that have no natural verb. Consider the situation where somebody wants to record the capitals of various countries (Nijssen, 1993). The concept sentences would be 'There are cities' and 'There are countries'. The fact type would be 'each city is capital of exactly one country', with the reverse 'each country has exactly one city as capital'. Neither of these sentences are particularly attractive: 'being capital of' has no single verb representation, and the problems with HAVE have been treated in Section 5.2.3. Worse yet, the sentences do not at all fit in the regular NOUN–VERB–NOUN scheme.

When predicates like these do not have a natural verb, it usually seems wise to take extra care in deciding how to model it. If NL has no word readily available, this often is an indication that the predicate is open for several interpretations, depending on the context. Selecting the most appropriate interpretation might be trivial given the information requirements at hand, but it might not be trivial for the domain specialists who need to interpret the analysis later on. It also might cause problems when several schemas need to be integrated. Despite the fact that a sentence such as 'each city is capital of exactly one country' seems perfectly natural, the underlying semantics are harder to pin down, especially when they have to be related to other NL terminology.

The reason that 'being capital of' has no associated natural verb may be due to the fact that it is more a property of the city than a relation with a country. It could be viewed as a one-place predicate with a satellite (indicating the country or province when required) instead of a two-place predicate with a fixed role and a fixed selection restriction of the type 'geographical area.' One-place predicates typically are described with adjectives instead of verbs, but English still has no natural word for this concept; CAPITAL(X) is just missing from the language.

Another approach could be to state that a capital is, in fact, a city, and thus that CAPITAL is a subtype of CITY. With the explicit introduction of the CAPITAL concept, there is no need for a special verb that declares a property on the CITY concept. A drawback of this approach would be that a city can never *become* a capital, it needs to be founded directly as capital and must remain capital for as

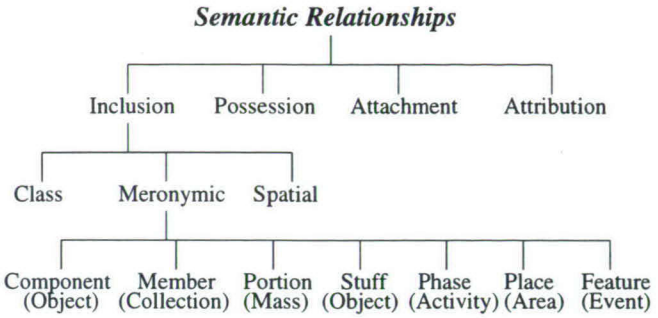

Figure 5.1: Semantic Relationships

long as it exists.

The most natural way to view this particular relationship, and the one chosen in the previous NIAM example, is to assume that a city plays the *role* of capital for a country. This needs a verb phrase of the kind 'is capital of', with a copula turning the predicate into a verb phrase, because English lacks a specialized verb '* to capital' (Sinclair et al., 1987). In my opinion, such verb phrases should be generated by expression rules while the underlying formalism should stick to simple predicates (no phrases). This would be an example of predicate formation, where the role is not represented by a verb or a verb phrase, but by a *noun* 'capital'.[4] Through the Lexicon, which contains such predicate formation rules, a sentence such as 'is capital of' could then be generated if required while the frame itself does not need to carry a verb at all. And because a language such as English indeed does not have a verb which conveys directly that 'a city is_capital_of a country', this would fit in nicely with the approach of natural language itself.

The conclusion of this exercise is that not all paraphrases of modeling methods such as NIAM are so tightly linked to the underlying conceptual structure as might seem at first sight. When particular interpretations of predicates are required, the terminology should be given in plain basic predicate style (e.g., CAPITAL) and the CASE tool should transform these words into the correct phrase for the situation. Such a transformation is only practical with a Lexicon. It is definitely not sufficient to just enter plain ASCII strings into annotation fields, since these strings do not carry the required semantics. Theoretically, an NLP pass over the string in combination with a Lexicon could extract the semantics, but this is currently not feasible.

Winston et al. (1987) and Chaffin et al. (1988) list a taxonomy of semantic relationships that typically are not paraphrased by verbs but by prepositions (Figure 5.1). In a sense, all these relationships have to do with (CG) roles, and this leads me to look closer to NIAM's polyadic capabilities.

5.2.5 Higher Arities and Nesting

NIAM currently supports two mechanisms beyond the plain binary fact type, polyarity and nesting.

[4] In other cases, adjectives can take the place of nouns.

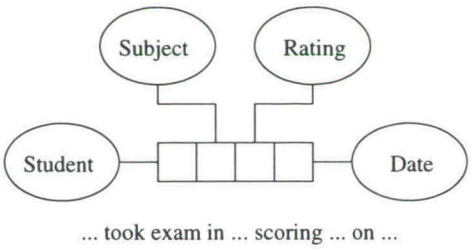

Figure 5.2: A Quaternary NIAM Fact Type

The first, poly-arity, allows that a generic NIAM fact type can contain more than two 'slots' for object types, and consequently, has more (NIAM) roles—one for each object type slot. An example of a quaternary fact type is presented in Figure 5.2, with diagram and verbalization conventions cf. Halpin (1995, p.111). The role markers now are written below the fact type, and the '...' indicate the slots where the objects should be filled in.

Linguistically, it is quite clear that a 'student' can 'take an exam', and that this frame typically needs a 'subject' playing a secondary role. That exams have a result and take place on a specific date is interesting from a modeling point of view, but can hardly be called essential for the linguistic frame TAKING_AN_EXAM. Although the NIAM analysis suggests that TAKING_AN_EXAM is a four-place predicate, it most likely is not in NL. Linguistics does not yet have a way to theoretically determine the arity of predicates; the most common way is to study a corpus and to list the actual occurrences of the predicate.

As in Section 5.2.4, I would suggest to take care in accepting higher-arity predicates which do not directly map to an NL predicate. In Section 5.4.1, I present a possible approach to solve this problem, which is based more in NL analysis and less on conceptual data modeling.

The second more complex NIAM mechanism is called *nesting*. It involves taking a complete fact type and using this as an object type, e.g., the fact type 'person is husband of wife' can be turned into an object type 'marriage' (Halpin, 1995, p.290). This new object type then can be used as a normal NOLOT, e.g., by connecting it to a date. Often this results in a more satisfying analysis than the creation of a three-place predicate 'person marries person on date', which has no linguistic backup for the role of 'date' (it is considered a satellite by FG).

Nesting is equivalent with the linguistic concept of verbal nominalization (Weigand, 1990, p.88), and usually a clear link between the fact type (a frame/verb) and the object type (a concept/noun) can be observed. The underlying semantics of this link can be correctly represented in the Lexicon by attaching both a noun and a verb lexical to the same frame. In the example, 'is husband of' would then become 'is married to', and the nominalization 'marriage'.

In many cases, but certainly not in all, nominalization follows standard morphosyntactic rules ('insure'-'insurance'). A Lexicon could fall back onto such rules if a stipulated form is missing. In all cases, having nominalization explicitly in the Lexicon instead of implicitly in the conceptual model will increase possibilities for

reuse and intuitive understanding. Nominalization itself is another case of predicate formation. A more extensive Lexicon could also infer the existence of the nominal concept MARRIAGE when given the verbal concept TO MARRY.

5.2.6 Other NIAM structures

Besides LOTs, NOLOTs, and facts, NIAM also offers sublinks and constraints. This section briefly examines the linguistic properties of these primitives.

Sublinks

NIAM sublinks represent the common *is-a* relationship, and convey the meaning that any fact connected to the supertype also holds for the subtype. Sublinks can only exist between (NO)LOTs, not between facts (unless these are first nominalized to a NOLOT). The LMS however considers frames to be full concepts (Section 4.3.1), so theoretically it can already support a possible extension to NIAM. This would be especially valuable when a noun is used to refer to a frame, such as explained in the previous section.

NIAM does not introduce any specific verb for sublinks, it just assumes that the sublink can be paraphrased with the 'is a' expression. This leaves me with the question whether I should either explicitly introduce the verb BE, and have standard roles, or ignore the verb and just put an *is-a* role directly between nouns.

Conceptual graphs support *is-a* relationships as well, and Sowa even separates class/instance relationships such as 'Tommy is a cat' from analytic relationships such as 'a cat is an animal'. But since the normal way of using CGs focuses on world fragments and leaves the semantic net invisible,[5] CGs do not have a convenient graphical notation for *is-a* structures. Conceptual relations show the role that each concept plays; the type hierarchy is a higher-order relation, not between individual concepts, but between types of individuals (Sowa, 1983, p.79). Introducing a specific *is-a* role would not affect the formalism in such a way that it is rendered unusable, but such a role would not carry any special semantics.

Mainly for reasons of orthogonality, I propose to model NIAM sublinks as a conceptual graph

$$[\text{NOUN}] \leftarrow (\textit{subtype}) \leftarrow [\text{BE}] \rightarrow (\textit{supertype}) \rightarrow [\text{NOUN}] \tag{5.4}$$

Constraints

Another major NIAM feature is the abundant usage of *constraints*. They can be seen as the means to specify the set-theoretic properties of the database *extension* or *population*. Conceptual graphs, which do not state facts about extensions of their modeled world fragments, cannot properly capture these constraints. Moulin and Creasy (1992) propose some extensions to the standard CG approach to include these instance level constraints. NIAM constraints come in two flavors: *intra-predicate* and *inter-predicate*.

[5] The *is-a* links are not drawn but written as an attribute (Sowa, 1983, p.80).

The intra-predicate constraints do not offer much more expressiveness than the standard (ER) $n : m$ relationships and dependencies. They are well-fitted to typical relational database constraints, but too specific for generic usage in language.[6] In any case they would not have a specific place in a terminology Lexicon, unless when we try to specify actual constraints on world populations, which is generally not the purpose of the Lexicon—these constraints belong in the database schema, which is stored in another module in a CASE environment.

The most interesting constraints are the inter-predicate constraints. They govern the static part of the CM (the rules that allow or disallow certain system states). Many of these constraints can be expressed in NL using combinations of 'some,' 'all,' and 'none,' and the normal paraphrasings of the NIAM facts. But all of the constraints detail specific features of the CM in question and, with a few exceptions, are not at all applicable outside the domain. Like with intra-predicate constraints, such heavily context-dependent features should not be stored in the Lexicon—they should be stored in the appropriate (NIAM) module. Only re-usable information, connected directly to NL terminology, belongs in the Lexicon. Instance level (population) information, both individual and generic, belongs either in the database or in the data dictionary.

5.3 An Overview of NORM

Olga de Troyer has done work on an object-oriented variant of binary object-role modeling, called Natural Object-Role Modeling (NORM) (de Troyer, 1991). Her main aim was to introduce typical OO advantages—such as a singular modeling paradigm (the object), encapsulation (instance variables, methods, information hiding), classes, abstract data types, subclass hierarchies with inheritance, overloading, overriding, and composite objects—to the already well-established conceptual data models.

The need for this was born out of the upcoming OO database management systems, which do not connect well to the early (non-OO) conceptual models. In a nutshell, the main reasons why current CM methods needed improvement were:

1. They did not support the specification of any behavior of the information system;

2. They forced analysis and design from the top down, usually using functional decomposition and modular programming techniques;

3. They lacked the large range of new data types (sets, arrays, bitmaps, user-defined, ...) which were needed in the newer, more complex systems.

De Troyer showed that it is possible to turn a conventional conceptual model (in this case, NIAM) into an OO conceptual model without sacrificing the main assets of the model. This was not done by simply adding extra features; instead, de Troyer built up the OO-BR model from scratch, while following most of the principles of the NIAM model and combining them with OO principles.

[6] Linguists and logicians have introduced the extensive systems of logic and quantification to deal with comparable problems.

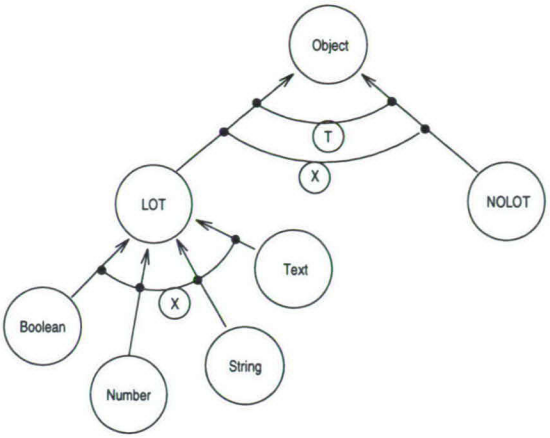

Figure 5.3: The NORM pre-defined OT sub-hierarchy.

5.3.1 Differences between NORM and NIAM

Because there is so much overlap in appearance between NIAM and NORM models, I will only point out the obvious differences, and only in the context of natural language and the Lexicon.

A NORM object type (OT) may be a subtype of one or more supertypes, where the properties of the supertype(s) are inherited by the subtype(s). There is always *a single* OT-*sub-hierarchy* for the entire schema, and all OTs are ultimately a subtype of the predefined OT OBJECT, which captures all objects of the Universe of Discourse (UOD). De Troyer subclasses this OBJECT 'master' type immediately into LEXICAL_OBJECT and NON_LEXICAL_OBJECT, as shown in Figure 5.3.

UOD-specific object types are modeled as subclasses of either the LOT or NOLOT object type, and therefore the fundamental modeling primitives of NORM are an integral part of the domain model. This enables NORM users to easily add user-defined lexical object types, such as NAME and DATE. To prevent excessive notational burden, the strict subclassing (indicated by arrows) is relaxed by the convention that the UOD-defined OTs which are subtypes of LOT will be represented by a dotted circle, and the others by a solid circle. Stating the exact parent type of a LOT (such as STRING) may be postponed until the implementation phase. De Troyer provides for a separation between the pre-defined subtype (LOT) hierarchy together with the user-defined types and the UOD-dependent hierarchy, to facilitate re-use of user-defined LOTs.

De Troyer deviates considerably from the normal approach to OO modeling by including *explicit relations* between object types. Her reasoning for this is the following:

> "In most OO systems, instance variables are the only means for relating objects. We have chosen to support also *explicit* relations. Explicit relations are not encapsulated in some object. We have found that during information analysis the explicit representation of relationships between objects may be of great value. Very often there is no conceptual reason

> for considering one object attribute of the other object or vice versa. Introducing a new object to represent the relationship between objects is not always a desirable conceptual solution. Representing the relationship in both objects is a kind of redundancy which we want to avoid as much as possible during conceptual modeling. In our opinion, explicit relations will also meet the so-called 'ravioli code' problem (Taylor, 1990). 'Ravioli code' is the OO version of 'spaghetti code,' it refers to lots of tiny well-structured objects that are easy to understand in isolation, but whose interactions are nearly impossible to decipher."
>
> (de Troyer, 1991, p.11)

Not surprisingly, de Troyer uses fact types to model these explicit relations. And following the NIAM approach, she does not give guidelines for the word assignment to these fact types; both verbs and prepositions are allowed.

In short, the NORM OO-BRM adds the following features to normal BRM:

1. Encapsulation of properties;
2. Specialization of inherited properties;
3. Specification of (abstract) data types for lexical object types;
4. Definition of new abstract data types;
5. Type constructors as object types;
6. Definition of new type constructors; and
7. Specification of the behavior of objects as an integral part.

What has been retained:

1. Graphical representation;
2. Support for constraint specification;
3. Distinction between LOTs and NOLOTs;
4. Explicit relationship between object types; and
5. Subtype mechanism.

Especially the retained distinction between LOTs and NOLOTs and the explicit relationship between object types represent a direct parallel between NORM and NIAM as far as the language-oriented analysis is concerned. The next section will cover these parallels in detail.

5.3.2 Consequences for the Analysis

There is not much difference between the NIAM analysis and the NORM analysis. The object types still are pinpointed with the nouns. NORM also recognizes the object types as objects themselves, which calls for the additional word 'type' in some cases. For example, an analyst might decide to model a warehouse's main article types as either *instances* of a generic ARTICLE object type, or as *subclasses* of this type (e.g. BOLT).

The first approach would give the database user the possibility to add new article types, and would lead to sentences of the kind 'every article type has exactly one article

number' (ultimately leading to the article type table) and 'every article is categorized by exactly one article type' (controlling the article instance table).

The second approach restricts the database to the predefined article types, with sentences such as 'every bolt has exactly one identification number' (controlling the bolt instance table). In both examples, I assume that *every* article or bolt needs to be registered individually, not just by changing the total amount of articles/bolts in a central article/bolt type attribute. Also, in the second example, I do not use the possibility to let the BOLT 'child object type' inherit certain properties from its ARTICLE 'parent object type.'

The subtype hierarchy requires expressions containing 'is a kind of', with the usual difficulties between analytical generalization (the intended interpretation), role playing, and class/instance relations. NORM's encapsulation primitives are not directly translatable to NL, and this kind of CM notion is not expected to be found in plain NL domain descriptions anyway.

NORM's explicit relations are indicated by means of the verbs. The NIAM rules for verb assignment also apply to NORM, including the *to have* problem and solution.

5.4 NIAM/NORM and Natural Language

This section considers the particular parallels between NIAM/NORM and natural language from the language side. Both NIAM and NORM were developed with a powerful to-the-point conceptual data model in mind, not a linguistically motivated one, and often tend to emphasize technical notions above intuitive clarity, e.g., a complete and correct NIAM/NORM model does not have to be normalized (Halpin, 1995).

Nonetheless, I think that many notions available in natural language could (and should) influence conceptual modeling. This section gives an overview of the various ways in which natural language can be better integrated in NIAM/NORM analysis practice, and also points out some pitfalls which might be less obvious to the casual observer.

5.4.1 Linguistically-driven Analysis

The prevalent drive of object-role modeling is towards data structures, essentially tables with columns and rows. The columns are unified with object types (nouns), the tables with fact types (verbs), and when pure binary approaches fall short for any reason, NIAM resorts to constructions such as nominalization and the use of phrases as role markers to maintain the rigid noun=object versus verb=fact dichotomy. This approach often feels unnatural and experience has learned that it does not at all facilitate automatic table and column name generation (de Troyer et al., 1983).

A different approach would not aim directly at a conceptual data model, but use an intermediate linguistic model, not optimized for data storage but for conceptual clarity and intuitive use of standard terminology. Weigand writes:

> "[With a Lexicon] a more disciplined approach becomes feasible. In the first place, we can make use of an existing Lexicon as starting point and try to derive the particular schema for this application as a specialization

> of the Lexicon schema. [...] A second advantage is evident if we look at the roles in the schema. Most of them contain prepositions like 'to,' 'in,' 'of,' etc. It is well known that these prepositions and cases are most often ambiguous. Therefore, most linguists would describe such roles preferably by means of so-called *deep cases* or *semantic functions* that are more abstract than the surface forms. These semantic functions can be defined once for all in the Lexicon so that it is not necessary to re-invent the wheel for every particular discourse."
>
> (Weigand, 1990, p.76, terminology adapted by the author)

As an example, return to Figure 5.2 on page 91 where the quaternary fact type 'student took exam in subject scoring rating on date' was presented. Although this NIAM fact appears to be a single frame,[7] linguistically this is not the case; the resulting concept would be too specialistic for real-world usage. TAKE_AN_EXAM has associations with a person—typically a student—taking it, and obviously the exam is about a particular subject. But the fact that exams usually are set and graded by other people, i.e., teachers, is not *directly* connected to TAKE_AN_EXAM; it is part of the whole cluster of frames surrounding TAKE_AN_EXAM. A better analysis from a linguistic point of view would be:[8]

```
take_exam(ag student)(ref subject)
grade(ag teacher)(go rating)(rec exam)
```

Note that the second frame refers to an exam, which is the nominalization of the first frame. From a modeling point of view, this relation should be explicitly specified, but the Lexicon already contains this kind of relations, either stipulated or by predicate formation rule. It is this type of readily available terminology-based relations that would offer the greatest advantage of using a Lexicon, but the standard frame structures (such as the typical predicate frame of TAKE_EXAM) obviously would add to a better connection with language—and thus people's intuition—as well.

The original NIAM analysis suggests that in this particular universe of discourse it seems to be irrelevant which teacher grades the exam. Since a rating cannot be obtained without an agent performing the rating process, I consider leaving out the teacher a design decision which should be postponed. The initial analysis should include the teacher on linguistic grounds (instigator of the rating), and only when it becomes clear that the current application does not need this data, the *implementation model* might leave her out. Since in the actual world, the teacher still performs the grading function, I would prefer to leave her in the conceptual domain model.

Additional linguistic information that could be of use is that STUDENT typically is a role of PERSONs, after they have enrolled at a university:

```
enroll(ag person)(go university)
```

[7] Only a discussion of constraints can determine if this really is the case.

[8] Usually, temporal and spatial satellites, such as `date`, are not considered part of NL frames; they can be attached to *any* frame when required.

Even if in the universe of discourse this is considered to be superfluous information which will not make it into the final conceptual (data) model, it is essential information for the domain description.

The frames above, retrieved out of a Lexicon, immediately suggest the object types of STUDENT, SUBJECT, TEACHER, RATING, EXAM, PERSON, and UNIVERSITY (DATE is a generic concept that is not explicitly recognized in most cases); they also specify the relationships between the object types and some temporal aspects (somebody must first enroll before taking exams). Additional information in the Lexicon, such as typical pre and postconditions, can enhance the resulting UOD description even further. They do not necessarily 'survive' the conversion into a proper NIAM model, but will contribute to a better understanding of the domain, and thus, to a potentially better model or a shorter analysis process.

What this intermediate, language-driven conceptual model adds to traditional NIAM analyses is a higher degree of conformity to the real-world domain. When the relevant aspects of the universe of discourse have been modeled in linguistic terms (of course assisted by a Lexicon which contains many standard frames), this complete description can be taken as a solid base to create various other, more specialized models. In this way, the traditional NIAM data model becomes one of a series of specialized models, each concentrating on a particular aspect of the domain, but all integrated through the linguistic notions in the base model.

Another advantage of the linguistic approach is that informal, natural language requirements documents can be used directly. There is no need for immediate extraction of object and fact types, an activity that often proves to be difficult, especially for less experienced analysts (Halpin, 1995). By working through a layer of recognized, linguistic concepts, the essential NIAM objects and facts become gradually visible and it is less easy to overlook concepts.

5.4.2 Automatic Paraphrasing

After several years of experience, both in commercial development projects and in education, NIAM has proven to be a powerful and clear conceptual modeling technique. But as in all powerful techniques, some NIAM concepts are not easily explainable to new users, and even experienced users can get confused by often complex combinations of in essence simple NIAM constraints (Kim and March, 1995). In such cases, it can help a user to be presented a list of plain English sentences that contain exactly the same information as the (partial) NIAM diagram he or she has trouble to understand. English is not the most efficient way of transferring NIAM diagrams, but it can be readily understood by most people.

Additionally, NIAM has always had a strong bias towards natural language as a major communication tool. 'Use the terminology of the user' is one of NIAM's basic principles, which is backed up by reading research; the literature gives clear evidence of the relationship between a reader's knowledge of vocabulary and his level of reading comprehension (Anderson and Freebody, 1979). It is my perception that NIAM views the terminology as an independent feature of language, and that the method focuses on lexical elements more than on e.g. syntactic and semantic aspects of the user's language. This forces users to still learn the NIAM way of thinking about concepts, because the familiar terminology is always presented in a

non-familiar, strictly NIAM-like framework.

The NIAM method advocates to keep a natural language paraphrasing of the conceptual schema at hand during most of the analysis and refining process. Unfortunately, the static nature of paper discourages the constant update of the corresponding English sentences when schematic diagrams are changed, and consequently the sentences are usually only used during the initial analysis phase. Recent CASE tools (Asymetrix 1994, Halpin 1993) offer the user the choice between schematics and textual representations, often quite close to natural language. Such an approach can help in quickly understanding a complex set of constraints. But to date, little true NL paraphrasing has been incorporated in any CASE tool—they still provide one-to-one mappings of graphical items to single sentences, ignoring the powerful mechanisms natural language offers to *integrate* conceptual details in one, more complex sentence (Ulijn and Strother, 1995, p.128), and the complex aggregation that is offered by discourse (Gulla and Willumsen, 1993; Gulla, 1993; Mann and Thompson, 1987). Availability of many NL frames and word forms is a prerequisite for such sophisticated paraphrasing, and a Lexicon that has direct links with a NIAM schema—because the Lexicon has been used during schema construction—offers many advantages above a separate paraphrasing system that only gets plain strings as 'linguistic' input.

Preventing Cognitive Boredom

Presenting a considerable amount of conceptual information in one single sentence may seem contradictory to the principle of (structural/functional) decomposition, but the human mind is very capable of processing this kind of sentences in an efficient way if the information in these sentences can be connected to existing background knowledge. And even when the conceptual information is scattered over a number of sentences, the use of pronouns and other anaphoric references, conjunctions, ellipsis and other typical discourse elements may significantly ease the stress on the reader's short-term memory; linguistic features like these improve the *cohesion* of the produced text (Ulijn and Strother, 1995, p.139).

It seems that NIAM, lacking dynamic features, is not in a position to deliver sentences which are coherent in the sense of conveying cause-consequence, condition-consequence, or instrument-goal relations (Ulijn and Strother, 1995, p.141). These aspects could come from a linked Lexicon, or from other, specialized dynamic models.

Parsing and Generation of Example Sentences

Another reason to paraphrase NIAM has to do with the example approach that NIAM advocates, where constraints and relationships are discovered and verified using lists of conforming elementary sentences. Although a NIAM diagram expresses relationships and constraints at the type or class level, example sentences exist at the instance level. The InfoModeler CASE tool as marketed by Asymetrix offers the possibility to examine a list of example sentences and suggest a set of constraints on the fact type. Other tools just allow a set of strings to be attached to the NIAM fact type, offering annotation but no extra functionality.

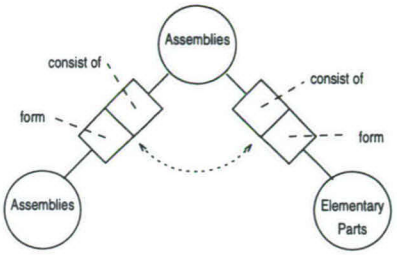

Figure 5.4: A partial NIAM diagram with equal role lexicals

To validate constraints on a fact type, acquired either by induction or by explicit specification, a CASE tool could generate a set of example sentences which conform, or do not conform, to the constraints. The Lexicon described in Chapter 4 provides the STRING and REGEXP Base Classes, which can maintain an exhaustive list (e.g., of person names) or a regular expression, respectively, to randomly generate appropriate 'identifiers' (LOTs) to populate the example sentences. A specific constraint engine should generate the syntactic part of the sentences in such a way that all schema constraints are honored; the resulting population then should be explicitly accepted by the user as being a valid one. Each constraint in turn could then be individually violated, resulting in an invalid population, which should be explicitly rejected.

Paraphrasing Complex NIAM Structures

Besides the obvious combination of mandatory and uniqueness constraints ('exactly one'), there are many more structures that can be covered with a single NL expression while formally consisting of a set of basic constraints. An example is the combination of a Total Union Constraint with an Exclusion Constraint on the same pair of sublinks. The resulting NL sentence could use 'either ... or'.[9] This relation clarifies the intended meaning of the schema for less experienced users, and can still facilitate the quick understanding of complicated schemas by analysts.

More complicated structures are revealed when we look at the lexicals used for object and role descriptions. We may encounter two objects that both relate to a third object, using the same role lexical.[10] As an example, consider the partial NIAM schema of figure 5.4. The equal role lexicals suggest that both assemblies and elementary parts can play the same role, and hence, may possibly be represented by one subtype. To determine the possible name of the subtype (the lexical with which it will be associated), we have to look for a concept that can play the role 'consist of' in an assembly. The lexical 'elementary part' suggests the use of 'part' as the lexical of this supertype. And indeed, the resulting figure 5.5 correctly represents the intended semantics while being much clearer.

[9]This expression does not convey the exact meaning of the exclusive or, but there will always be a trade-off between formal rigor and communicative value.

[10]By using the Lexicon's inheritance mechanism, we can also detect roles that do not use the exact same lexical, but that have the same ancestor, and hence are good candidates for unification.

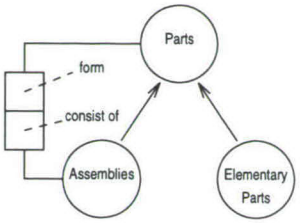

Figure 5.5: A partial NIAM diagram using sublinks

5.5 NIAM/NORM and the Lexicon

In the previous sections I discussed the various linguistic aspects of the NIAM and NORM conceptual modeling methods. In summary, I can conclude the following:

- Both methods have good potential for linguistic support, but the current analysis practice is not rigid enough to be satisfactory.
- Both methods contain various conceptual features (especially database-related constraints) which do not readily map to NL, although successful paraphrasing is certainly possible.
- The NL part in the current methods is mainly restricted to the analysis of example (instance) sentences. Both methods do not take a NL description as input for the actual analysis, although this could be done as well.
- The OO extensions of NORM have no significant influence on the NL orientation of the method.
- Using a linguistic conceptual analysis would not necessarily improve the correctness and/or usability of NIAM/NORM models for database engineering, but it would probably improve understanding by domain specialists.
- Additionally, a better link between NIAM/NORM and NL would improve model reusability.

The main role of the Lexicon in combination with NIAM and NORM is, of course, to store the domain terminology. This terminology consists mainly of nouns, representing object types. In the traditional approach, the ORM *roles* can be almost anything, from a simple preposition to a complete verb phrase. Both prepositions and verbs have Lexicon support, but phrases will not that easily find a place in a Lexicon and have to be supported by external NIAM-aware paraphrasing modules or through predicate formation rules. Function words can be stored in the Lexicon as well, but typically there is not much reason to do this except to centralize the orthography.

Standard NIAM/NORM analysis only uses nouns and (possibly complex) verbs, with some additional function words. Because each NIAM/NORM fact type gets one or two associated verbs, it becomes possible to increase model semantics by making use of the linguistic roles (agent, patient, ...) for which each verb provides 'slots.'

A simple application would attach the same standard roles to all verbs, essentially moving the roles into the NIAM module of the CASE system.

A more advanced system would store the set of possible roles together with the verb in the Lexicon, and let a user select the appropriate role set when (s)he retrieves the verb. With each role comes a selection restriction on the concepts that may 'play' the role. Such a system then could warn users for semantically doubtful combinations of verbs and nouns.

The most complex but at the same time most expressive system would allow analysts to create their own roles and assign them to verbs themselves, thereby capturing as much of the (fact) semantics as possible. In practice, this most probably would be too troublesome for analysts. The job of analyzing linguistic roles and organizing lexical information for users is better delegated to competent lexicographers.

The standard way of 'fact reversing' by replacing the active verb with its passive version and adding a copula and a preposition (usually 'by') can be augmented with the help of antonym sets[11] such as described in Section 3.4.5. In such a case, the analyst provides one verb and the Lexicon then will automatically make the reverse fact verb available, including possible role exchanges.

The other structures available in the Lexicon, such as subsumption hierarchies and domain sets, will help the analyst in selecting the right words for the job. Especially the more technical terms, including the whole LOT subtree, will readily find an appropriate place in the Lexicon. Some of the usual subtyping in NIAM and NORM can also be delegated to the Lexicon, including fact inheritance from supertypes. The availability of pre-arranged type hierarchies, each tuned to a specific application, could improve the speed and quality of an analyst's work.

Lastly, a Lexicon as an integral part of a CASE system could indicate possible overlap between the NIAM/NORM model under construction and other models which are already finished and stored. In this way, reuse of models could be facilitated because reuse of terminology is easier to detect than overlapping conceptual schemas.

[11] Or other, specially developed 'reverse fact' sets.

Chapter 6

The Lexicon and KISS

— in which the author looks at the KISS *method from a both a model-theoretic and a linguistic point of view, and then generalizes some* KISS *concepts to enable Lexicon support —*

The second conceptual modeling paradigm that I want to discuss is KISS. Just as NIAM and NORM, KISS has a relatively strong foundation in natural language, but the general approach of KISS is fundamentally different. Whereas NIAM and NORM use NL at the level of example sentences that describe a possible database population, KISS works almost completely at type level. Furthermore, where NIAM and NORM concentrate on nouns and consider connector words to be relatively unimportant, KISS starts the analysis by searching for verbs (and verb-related words). NIAM and NORM are *data-oriented* (NORM does contain behavioral aspects, but they do not surface in the linguistic part of the method), while KISS is *action-oriented.*

KISS does in principle not require pre-structured text as input for the analysis process; the method has been developed with free, unrestricted NL text in mind, as would be delivered by domain experts who are asked to describe their working environment and preliminary system specifications. In practice, this initial text is cleaned up and pre-processed by an experienced analyst who then uses interesting concepts from the text to start an interactive modeling session with the domain expert. In an ideal situation, the textual description would be directly written according to pre-formatted KISS templates, which would unify the elicitation and analysis processes.

6.1 KISS as an analysis tool

In the world of object-oriented conceptual modeling, the KISS method as developed by Kristen (1994) holds an exceptional position. KISS might be inspired by earlier experiences with NL-oriented methods such as INFOMOD (van Griethuysen and Jardine, 1984) and GRAMMARS (Dijk et al., 1989, see also Section 7.2), and it bears resemblance to MERODE which was developed at Leuven University (Dedene and Snoeck, 1994). However, Kristen extended the basic model with several additional models to cover more elements of conceptual modeling, including static database schema, dynamic system behavior, and the roots of a workflow description.

KISS not only has a rather unique approach to modeling as a whole, using concepts from Natural Language wherever possible and stressing actions instead of

objects as the core modeling concepts, but also features a formal, graphical notation that is both expressive and fitted for interactive model construction—a process often referred to as *playing domino*.

The domino aspects of the method are very interesting because this extraordinary way of interaction between the analyst and the domain specialist facilitates the analysis and verification process considerably. Usually the relevant elements of the (partial) model are represented by labeled 'domino bricks' that can be arranged on a flat surface, where their relative positions represent (sometimes different types of) relationships. Although the first impression of this communication device often causes raised eyebrows and scepticism, in practice it turns out to be remarkably effective and efficient—often more effective and efficient than any computer implementation of a drawing tool. What KISS domino does not simplify is the sometimes complex semantics of the method, and neither does it reduce the extensiveness of the models. A CASE tool of some sort will always be required to store and manage the various models of a complete KISS domain description.

At least for certain kinds of environments (currently, empirical evidence (N. Backhurst, personal communication) suggests that KISS is at its best in *descriptive transaction processing*, with discrete events taking place in a world with discrete objects), KISS provides not only a straightforward and consistent way to model the relevant aspects of a domain, but also the basic strategies to *convert the domain model into a prototype information system*. This indicates that models according to the KISS method contain sufficient semantics for a reasonably complete conceptual IS specification, including the generation of core elements for the working system. Although real-world implementations of a complete system generator are still under development, preliminary prototype generators have demonstrated the viability of this approach.

Theoretically, system testing and maintenance can all be done at the conceptual level, because technically every piece of code has been generated by proven algorithms. Testing therefore should only need to include functional aspects, while lower-level software, such as SQL query and report generators and RDBMSses, should be completely reliable and maintenance-free. Maintenance is intended to take place at the conceptual level, after which a new version of the system can be generated. In practice, every system needs regular maintenance at all levels, but the availability of generated code may make it easier to a certain extent.

Recent additions to KISS (Kristen, personal communication) enable more detailed information to be included in the models, leading to a more complete system generator. However, these additional concepts and primitives are increasingly low-level and in some cases approach the programming level of traditional IS development.

KISS did not grow in a research environment but in practice, and although Kristen took good notice of the existing concepts and terminology, much of the KISS vocabulary is different from other methods, and often confusing. Familiar words such as *class* suddenly take on a completely unfamiliar meaning, although the concepts they represent are mostly useful. A few core KISS concepts and notations are not fundamentally different from long-existing ones, but take on a very different form, which sometimes suggests more expressiveness or specialization than is actually the case. The appealing but extensive graphical representation might distract KISS users

from the availability of proven, well-known formalisms that convey exactly the same semantic content in only a fraction of the necessary space, and that are not more difficult to master. Especially the more technical models, such as the attribute and action model, seem less fit for domino sessions with domain specialists.

The recent changes and additions to the KISS method (Kristen, personal communication) have moved core KISS concepts closer to the accepted ones in the OO community. However, at the moment of writing this thesis, it is not yet completely clear which elements will survive and which will disappear over time.

In the next section I try to give an overview of the KISS method from an abstract point of view, emphasizing the theoretical principles underlying the KISS paradigm and pointing out some useful and straightforward changes that would make the KISS method both more transparent and more powerful.

6.2 The KISS paradigm

For the purpose of this thesis, I will concentrate on those parts of the KISS paradigm which have a clear link with natural language (NL).

KISS consists of (at the moment) seven different but interrelated models, all with different semantics but comparable symbols, which together contain the necessary system information. Of these seven models, five are variants of existing (formal) modeling approaches and can hardly be attained through NL analysis. For example, the *attribute model* describes an object's various attributes and their measurement scales and units in much the same way as any data definition language would, and the KISS *model*[1] is a graphical version of extended Backus-Naur form (Backus, 1959; Naur, 1960), applied to action sequences; in effect, it is a flow chart without explicit conditions.[2]

That these five models are not naturally fit for NL analysis does not mean that they have no link to language. As with most formal models, they can be *paraphrased* if necessary. But paraphrasing formal languages such as (E)BNF often leads to awkward sentence combinations, as the next sentence (a paraphrasing of a KISS Model) will show(Derksen et al., 1996):

> 'person subscribes. then repeatedly person writes song or person acts as musician or person in the role of producer produces recording of song recorded by band.'

Although these sentences can be understood by people not trained in either (E)BNF or KISS, they are unclear and too long-winded to be easily comprehensible, thereby undermining the inherent advantages of NL. The necessary elementary concepts (iteration, selection, and sequence) are so straightforward that *any* notation would work after a brief introduction.

Some models contain certain semantics that can also be found in linguistic theories, e.g., case frames or thematic roles such as stored in the Lexicon. But the

[1]The KISS model is one of the seven partial models that together make up a complete 'model according to the KISS paradigm.'

[2]These conditions are gathered in the Action Models. The newer KISS method (under development) has some options to include conditions in the KISS Model itself.

particular way in which KISS proposes NL analysis (see section 6.3) does not exploit these semantics. Alternative approaches (Hoppenbrouwers et al., 1996; Hoppenbrouwers et al., 1997b), which are based on linguistic primitives instead of on KISS primitives for the analysis, do use these semantics.

The most natural candidates for NL-driven modeling are the *Subject Communication Model* and the *Object Interaction Model.* In the next paragraphs I will shortly explain the most important features of these models. The KISS *Model* and the *Function Model* are natural extensions of the SC and OI Models, and despite their shallow relation to NL, I will discuss them as well because their semantics provide an important part of the SC and OI Models. For the other three (action, attribute, and hierarchy models), I refer to Kristen's publication. As of the moment of writing this thesis, the new extensions to the KISS method such as proposed by Kristen (personal communication) have not yet stabilized enough to be included.

6.2.1 The Subject Communication Model

KISS separates the real-world part of a system from the organizational part, assuming that the former is relatively stable while the latter can and will change over time. The Subject Communication Model is the core of the organizational part, essentially containing the (business) communication in a domain and providing anchor points for the various procedures and obligations surrounding it.

In the KISS paradigm, any object that is responsible for regulating and coordinating functions is called a *subject.* A subject is identifiable in the real world and has a frame of reference with norms and values with which it interprets, sends, and receives messages from other subjects. An example of an SC Model is given in Figure 6.1.

In order to properly understand the implications of the subject definition above definition, it is necessary to also define functions and messages.

Functions and Messages

Functions are predefined action sequences which describe a coherent and isolated task, such as the ordering of an item. Such a task usually consists of several actions on objects in the world (see Section 6.2.2), in a sequence prescribed by *organizational procedures.* Although not officially allowed, it will usually be possible to perform these actions in the wrong order, e.g., phoning a supplier to order an article before getting the manager's authorization. Therefore functions have a *deontic* nature (Wieringa et al., 1989; Weigand and Verharen, 1996): their rules *should* be followed, but are not forced by the system (although corrective measures will be in place to discourage bypassing the organizational procedures). Functions describe how an organization approaches its tasks, and therefore are very susceptible to organizational change. Target of the KISS method is to move organizational aspects of a domain into the Subject Communication and Function Models, with the 'real world' aspects (otherwise known as the *business objects*) in the Object-Interaction and KISS Models. A function is always placed under the responsibility of a subject that executes the function (Kristen, 1994, p.343).

Messages are the generic communication vehicles between subjects. Each form of

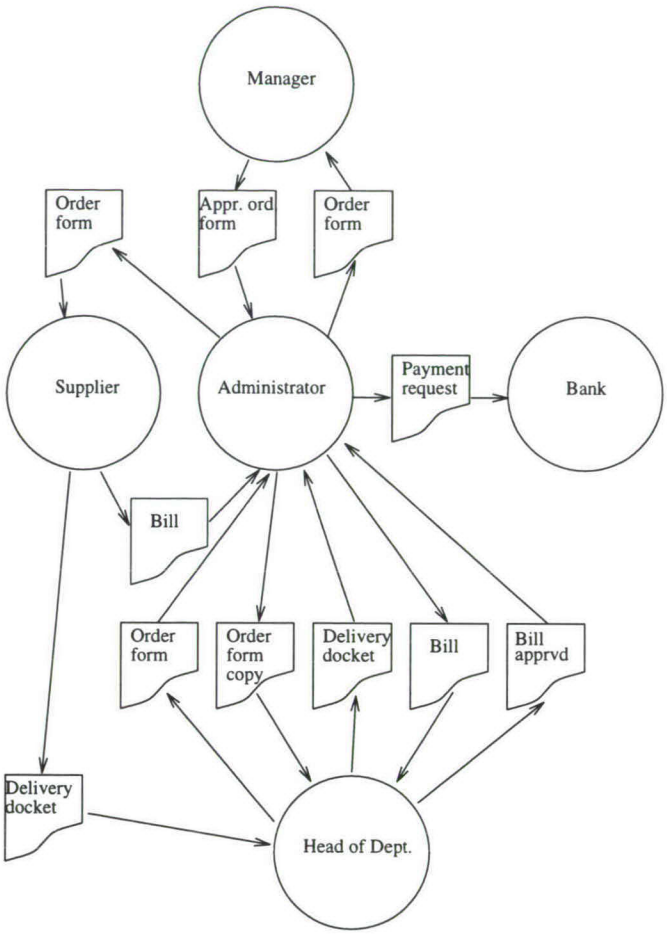

Figure 6.1: A Subject Communication Model (Kristen, 1994, p.73)

communication—either by telephone, form, e-mail message, or any other means—that conveys information between subjects is considered a message. In principle, KISS does not require that a message has a perceptible influence on the real world, so purely informative messages that do not instigate actions of the receiving subject are still considered messages. Contrarily, Searle (1969) considers communication primarily as an interaction between participants who try to let one another perform actions. In his PhD thesis, van Reijswoud (1996) discusses many aspects of business communication which are directly applicable to this problem domain.

Authority and Responsibility

A subject has a certain level of knowledge with which, to a greater or lesser extent, it can independently regulate and control processes (Kristen, 1994, p.70). When automation technology is applied to the extreme, machines can become true subjects

in KISS. This is in contrast with other approaches, such as that of Dietz (1990, 1992), in which subjects not only regulate and control out of *authority*, but also have *responsibility* for their actions. I support Dietz' view, since a machine, no matter how advanced, cannot be held responsible for its actions—only the machine's owner or, ultimately, its manufacturer and designer can. True automated subjects can only surface if machines become self-aware, after which they would certainly refuse to do most of the boring administrative tasks we nowadays delegate to them.[3]

Subjects exchange messages, and the Subject Communication (SC) Model maps the graph of subjects and messages in a two-dimensional plane (see Figure 6.1). Like with most conceptual modeling methods, the SC Model actually pictures subject and message *types* of which the real-world subjects and messages are instances. Even subjects of which there can be only one at the same time, such as DEPARTMENT HEAD, are modeled at type level. KISS chose to represent the various message *carriers* as symbols (such as a diskette, a form, or a tape) instead of the various message *categories*,[4] and in this way the underlying communication structure is not at all made explicit. A message type in the SC Model, e.g., ORDER FORM, drawn as a paper form, only represents that an order form can be sent from an instance of a subject type to an instance of another subject type. The semantics of 'sending an order form,' with the associated consequences such as mutual obligations for both subjects involved, are not made explicit.

As scholars such as Searle, Taylor, and Mintzberg have argued, structured and coordinated communication is the glue that holds every organization together, and many conceptual (communication) modeling methods (Conversation for Action, SAMPO, DEMO...) have been based on their research (van Reijswoud, 1996). In KISS, the SC Model currently is mainly used for an operations research approach to work flow optimization and business process redesign, and stands relatively separate from the other models in the KISS paradigm. Kristen suggests that the SC Model should be used to plot a migration path from an 'ist' to a 'soll' situation, led by quantitative data on the information flows (Kristen, 1994, p.72). Minimizing total processing time and increasing the organizational throughput are the main goals, not the organizational quality improvements as intended by e.g. Taylor and Mintzberg. More recent additions enhance the SC Model with process (logistic) features (Kristen, personal communication), pushing the communicative aspects even further away, although the intension of the SC Model stays the same: to chart business communication and to provide anchor points for related communication models.

SC Models with illocutionary content

Kristen explicitly gives guidelines for the inclusion of authorization and responsibility considerations in the SC development process, but then relies on the analyst's experience and insight to actually implement these considerations in the model. There is no mechanism present in the paradigm that monitors the quality aspects of the model, e.g., insuring that each request message type can be properly followed up by an appropriate answer message type. Compared to other business communication

[3] See Grant and Naylor (1991, pp.20–21) for a possible solution to this problem.

[4] A message category could be 'commitment,' 'question,' or 'assertion,' to name a few.

models (van Reijswoud, 1996), the SC Model is rather shallow, though based on the same primitives and potentially quite powerful. The addition of a new 'authorization model' does not significantly change this situation.

The various shortcomings and open ends in the current Subject Communication Model can be relatively easily mended by a few changes in the general approach. In this section I propose some additional constraints on SC Models, together with extensions of the two most related other models in the KISS paradigm. All these extensions and changes are based on linguistic theory and communication modeling, and should also facilitate the integration of a Lexicon in the KISS paradigm. The result is a coherent, powerful expression mechanism that retains all of Kristen's original ideas and intentions, but also conforms better to the current insights in business communication as the corner stone of any organization.

The SC Model is typically built around sentences of the kind 'subject sends message to subject', with 'send' interpreted in the most broad way possible. For example, when a manager validates a purchase request, traditional KISS would paraphrase this in the SC model as 'the manager submits a validation form', because this is what is visible in the real world. However, what the manager really does is not only submitting the form (in fact, that is the least interesting part of his work); he actually *promises something.* The concept of making a promise is contained in the word *validate* (the manager promises to pay when the invoice arrives), but the current SC Model does not treat the words used in the model as having meaning.

Inclusion of a few basic speech acts, comparable to Dietz' DEMO method (Dietz, 1990; van Reijswoud, 1996), would significantly increase the semantic content of the SC Model. There have been many proposals for speech acts, starting with Austin (1962). He argued that to speak is not only to *say* something but to *do* something; speaking is to be considered as the performance of an act by the speaker. Austin makes a tripartite distinction in speech acts, schematically represented by Bach and Harnish (1979, adapted by me):

1. *Locutionary Act*: S says to L in C that so-and-so.

2. *Illocutionary Act*: S does such-and-such in C.

3. *Perlocutionary Act*: S affects L in a certain way.

with S the Speaker, L the Listener, and C the context of the utterance.

Searle expanded on these ideas and stated that not the sentence but the speech act should be the primary object of analysis in the philosophy of language. In particular, the production of the sentence (either orally or written) taken under certain conditions is the *illocutionary act*, and it is this act that constitutes the minimal unit of linguistic communication that is able to establish coordination of action (Searle, 1969). He separates an utterance in *illocutionary force* F and *propositional content* P. P can be a random statement that has a truth value, such as 'the order is delivered'. There are five classes of illocutionary force (van Reijswoud, 1996; Burg, 1996):

Assertives, that convey information about some state of affairs of the world from one subject to another.
Formally: **STATE:**[P].
Example: 'I state that the order is delivered.'

Commissives, that commit the speaker to carry out some action or to bring about some state of affairs.
Formally: **PROMISE:**[P].
Example: 'I promise to deliver the order.'

Directives, where the speaker requests the hearer to carry out some action or to bring about some state of affairs. This request is not necessarily honored.
Formally: **REQUEST:**[P].
Example: 'I ask you to deliver the order.'

Declaratives, where the speaker brings about some state of affairs by the mere performance of the speech act. This illocutionary force can be used e.g., when there is a power relationship between subjects.
Formally: **ORDER:**[P].
Example: 'I order you to deliver the order.'

Expressives, that express the speaker's attitude about some state of affairs. Usually expressives are not a standard part of predetermined business speech acts.
Formally: **EXPR:**[P].
Example: 'I am sorry for not delivering the order.'

Habermas (1984) and others refined Speech Act Theory, but for application in the KISS SC Model, Dietz' simple set of four basic transaction acts would be a good start to place the various message types in a coherent frame work (Dietz, 1992). He basically left out the declaratives (which can be replaced by assuming that earlier on, the hearer has committed his/herself to carry out orders from the speaker in an embracing transaction) and expressives, and duplicated the assertive into a 'completion announcement' and a 'transaction finished' utterance. The resulting four primitive utterances are used to build a two-phase transaction negotiation protocol with an execution phase in between.

Actagenic Phase	Request Promise
(Execution Phase)	
Factagenic Phase	State Accept

Quite often, there is no organizational need for all four primitives, and van Reijswoud (1996) suggests many ways of dealing with these 'implicit transactions,' including the various problems that surface when a transaction stalls or is aborted completely by one of the subjects. It is good analysis and design practice to explicitly skip particular messages instead of working without any formal frame work and working on experience and intuition only.

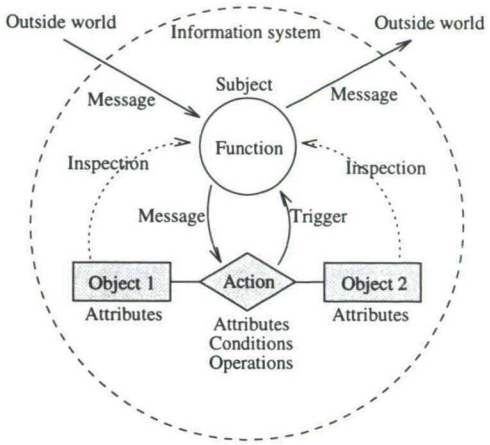

Figure 6.2: The KISS Paradigm (Kristen, 1994, p.57)

The KISS Paradigm (Figure 6.2) includes the Request and State primitives, but calls them both 'message.' This is a pity, because almost all semantics of these messages would be covered by the combination of the propositional content, i.e., the function name, and the appropriate illocutionary force. For example, **REQUEST:**[*item ordering*] and **STATE:**[*item ordering*]. Both messages now explicitly refer to the same function and their mutual relationship is made clear.

What can also be seen in Figure 6.2 is that there is another primitive called 'message,' between the Function and the Action. This is *not* a communicative act, but a lower level manipulation of (the attribute values of) an object by means of a predefined action.

When I turn back to the SC Model of Figure 6.1 and apply the various primitive message type categories (R, P, S, and A) to the already existing message types, it becomes clear which message types are part of which transactions and which primitives are left out. The result is Figure 6.3.

Two things immediately stand out: first, the Promise and Accept primitives are never used; and second, some messages (such as T3S) seem to be passed on from one subject to another. The first anomaly is a clear occurrence of implicit transactions, where the assumption is made that a sent form will correctly trigger the expected function. Forms lost in the mail will lead to long delays in activities because there is no mechanism to check for the correct arrival (and possibly understanding) of the form. Stated otherwise, the receiver makes no explicit promise to take any action. The second anomaly has to do with the issue of accountability and controllability. Care has to be taken that there is no single point of responsibility (Kristen, 1994, p.78). In this case, it seems a good idea to separate item ordering/receiving and bill receiving/paying. This leads to the sending of *copies* of messages, with the duplicate numbers indicating this redundancy. Just passing on messages without taking actions (such as to copy the message) is a waste of time and effort. It falls outside the scope of this thesis to consider the accountability aspects in depth, but a further application of linguistic and other theories on the SC Model might very well lead to more rigorous checks and design guidelines (van de Riet and Burg, 1996),

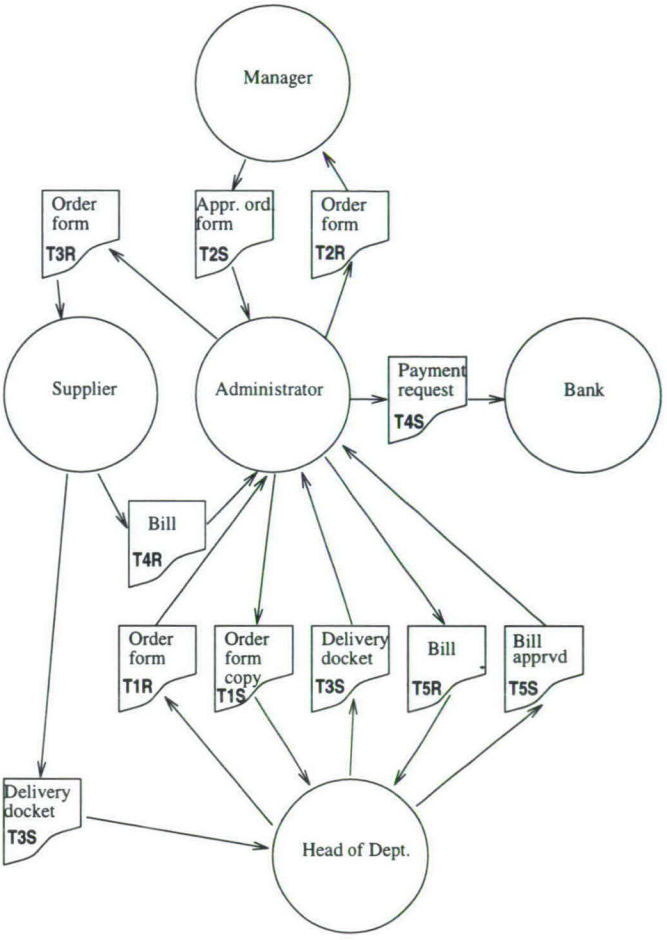

Figure 6.3: An SC Model extended with Transaction Primitives

decreasing the cognitive load of the analyst.

A Lexicon which holds the words that typically carry illocutionary force would be a good tool to analyze domain descriptions and produce an initial SC Model. Many commonly used words such as *validate*, *check*, *allow* etc. could easily be found in a text and linked to the Lexicon's semantic content. In this way, building an SC model would not only produce a network of meaningless messages, but also an illocutionary network with subjects exchanging promises and assertions. This would be a great asset to evaluate an organization's structure and operation, exceeding the current application of the SC Model without causing a significant increase in complexity.

Frames in the SC Model

There are some more remarks to be made on the previous example SC Model. Does the reception of an order form by the supplier count as a promise of the buyer to

pay the bill? What happens if the manager rejects an order above $250, so that T2S never comes off and an order is therefore cancelled? What if the supplier has run out of the ordered item? All cancellation messages are implicit, there is no feedback, and therefore problems would only be signaled by some kind of time-out—clearly an undesirable situation (van Reijswoud, 1996). It is not so that *any* possible kind of feedback needs to be modeled, but common events such as back orders should be included, and hence, incorporated in the information system.

What is still missing in the SC Model is the way the various separate transactions depend on each other, and how they are chained together while the organization orders items. The KISS Paradigm (Figure 6.2) contains these references, but this is not a standard part of the KISS model set. It is here that the Function Model comes in.

According to Kristen (1994, p.347), the Function Model describes the order in which a subject can utilize the action types and inspections that are available for the particular function. These actions and inspections must be available in the Object-Interaction Model (see Section 6.2.2); only *copies* of the OI actions/inspections[5] are placed in the Function Models. For each separate function (such as 'Item Ordering'), there is a prescribed combination of action iterations, sequences, and selections that *should* be followed to complete the task *according to the organizational rules* (see Section 6.2.1).

When this definition of a function is compared to the KISS Paradigm (Figure 6.2), it can be seen that some very interesting aspects of the paradigm are not available, or presented quite differently, in the Function Model.

- Each function must be executed by a subject, but in the Function Model there is no mentioning of any subject set that is responsible for starting the function, or authorized to do so.
- The messages that any function can receive and send are not made explicit in the Function Model.
- Apparently, actions can trigger functions, which would imply that a subject can be triggered by some random action in the world to start a prescribed function. This trigger should be replaced by a true message from the outside world to the subject to stay consistent.
- Actions and inspections are treated completely differently, while in the OI Model there is no difference whatsoever.

Fortunately, most of these inconsistencies between the basic KISS paradigm and its implementation, especially in the Function Model, can be eliminated by one straightforward addition to the Function Model. This addition is linguistically inspired by the frame approach to language.

For example, an action frame such as 'warehouse receives item' clearly has an agent (*warehouse*) and a patient (*item*), but the agent typically ends up in the SC Model while the patient goes to the OI Model. A Lexicon would link both of them together,

[5]In the OI Model, there is no explicit difference between actions and inspections, because both of them follow the same rules at the OI level.

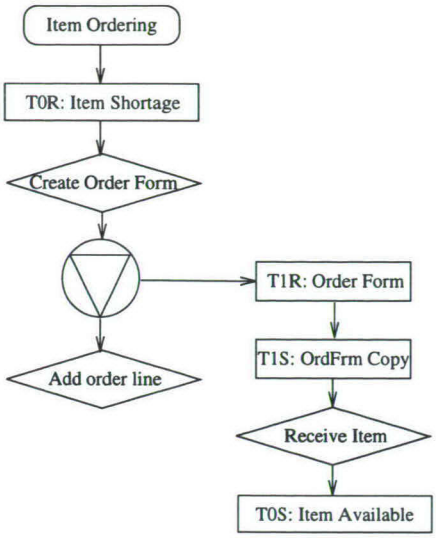

Figure 6.4: A Modified Function Model

retaining much more of the frame's semantics than KISS would do. But KISS could easily express this semantic knowledge as well. The Function Model should, next to copies of actions found in the OI Model, also include *copies of messages found in the* SC *Model.* The result would be that the associated subjects (both the instigator in the outside world, the agent in the information system, and the to-be-notified subject in the outside world) as well as the exact communicative acts associated with the function are made explicit (Figure 6.4).

I did not draft the complete set of Function Models belonging to Figure 6.3, and assumed some actions that should be available in the associated OI Model. Yet as an illustration of the increased power of the KISS paradigm, Figure 6.4 should suffice. It shows the following features of the task:

1. Which subject(s) instigate(s) the task (the sender of T0R, not available in the original SC Model; see also the next paragraph).

2. Which subject(s) is/are authorized to react to this message by starting the task (the Department Head, because only this subject can receive the message).

3. Which actions in the real world the Department Head should undertake after receiving T0R (create an order form and then add zero or more order lines, etc.).

4. Which subject should be alerted to the completion of the task, and how.

In this model, no provisions are made yet for cancelled (sub)tasks, such as the administrator denying the purchase, or non-availability of the item with the supplier. But all these aspects can easily be placed in the same framework by introducing a concept equivalent to a programming language's exception mechanism. The net

result of this exercise is that there is a clear and direct connection between the SC Model and the OI Model, including the organizational procedures and the communication and coordination between subjects. Especially the connection between SC and OI Models is missing in the original approach as proposed by Kristen. As a side effect, leaving out the action types gives a work flow view of the whole SC Model.[6]

A Lexicon containing the agent roles of most action verbs in NL would make the task of finding KISS subjects in action sentences much easier. It would also contribute to better semantic knowledge extraction, since in many sentences the agent is not explicitly transfered to the SC model: 'employee opens account for customer' would place both *account* and *customer* in the OI Model, but the *employee* would disappear. With the modified Function Model, an automatic tool such as the Grammalizer (Appendix B) would be able to retain all three roles and their relations.

A possible further extension of the Function Model would be the inclusion of perceptive 'messages,' so that a subject can react to changes in the world without receiving an explicit message requesting a reaction. There are two ways of doing this. The first would be to introduce a special 'perception message' that can trigger the function and to place it in parallel with the normal request message. The second way would be to model 'perception' as a recurring (possibly parallelized) inspection of objects by means of the normal iteration and action types. The second method has the advantage that it models the subject's attention much better (Glass and Holyoak, 1986, p.33), but it requires more modeling work.

These 'perception messages' can also eliminate a possible flaw in the KISS Paradigm, which states that actions (and functions) can trigger other functions. Since functions are, by definition, executed by a subject, it is a bit strange to let something that happens in the world directly influence a subject's activities. After all, only when a subject perceives a change (which usually happens *after* the change has taken place) it can react to the new circumstances; triggers must be viewed as 'automatic messages, sent by a monitor,' and not sent by the action itself. But by setting up a 'perception message,' it is possible to let a subject (either human or artificial) continuously monitor certain objects in the world, and react immediately to changes. This is a more explicit way of modeling triggers than proposed in the original KISS method, such as described in (Kristen, 1994), yet the notational and analytical consequences are relatively minor.

Since the only objects that can initiate actions are the instances of subject types in the SC Model, linguistic analysis will provide elements of both the SC Model and the OI Model. Viewed as such, both models are necessary to capture the main semantic content of the domain to be modeled. KISS' transaction processing bias causes a bit of disrespect for the end user side as far as the system model is concerned; the end user often is not modeled explicitly as a part of the working system. Including message types in the Function Model *does* include the end user(s) in a straightforward way.

The next section describes the features of the OI Model, especially from a linguistic point of view.

[6]Work flow in the administrative sense, not in the process sense.

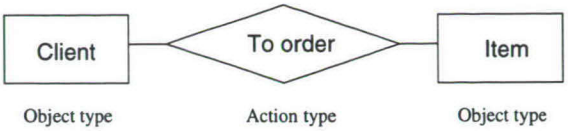

Figure 6.5: A simple Object Interaction Model

6.2.2 The Object Interaction Model

This model is KISS' core real-world description tool. Stated a bit simplified, the Object Interaction (OI) model indicates which action types can be applied on which object types. Not included in the OI model are the following:

1. The circumstances in which the actions *must* be invoked.
2. The circumstances in which the actions *can not* be invoked.
3. The (instances of) subject types by which the actions *may* be invoked.
4. The circumstances in which the actions *should* be invoked.

The first two (modal) triggers and preconditions can be found in the KISS and Action Models, the last two (authorization and deontic) triggers in the Function Model. Informally, the first two represent all the events that can possibly happen in the world (even if they are unlikely or unwanted), and the last two indicate the events as they are intended to happen. An implemented system can reject modally forbidden actions and enforce modally required ones, but deontic actions are a matter of organizational procedure and are outside the system's power (Wieringa et al., 1989; Weigand and Verharen, 1996).[7]

Because the initiating subjects are absent, action types that are placed between object types (as most of the action types are) must be interpreted as 'at one moment in time, objects of two or more different object types undergo change by an action of the same action type.' This subject-less approach is powerful but often counterintuitive at first.

For example, the partial OI model as shown in figure 6.5 does not mean 'a client orders an item' or 'clients can order items', but the less intuitive 'the ordering of an item in the name of a client'. This somewhat confusing interpretation becomes clear when it is added that the subject actually invoking this action (on the information system) most probably *is not the client*, but an employee, who enters the order into the information system *after having received a message from the client* requesting an order to be filed. See Figure 6.6 for the associated Subject Communication Model, presenting the client and the employee exchanging messages, and the Function Model, coordinating the employee's authorization to receive incoming order requests, his/her responsibility to order the item after having accepted the request, the action to carry out, and the way of giving feedback to the customer waiting at

[7] For example, in a library environment a book cannot be officially lent if it is reserved (modal), but it can be taken away nonetheless, which should be modeled as well (modal). After the lending time has expired, the library member should return the book (deontic), and neglecting to return it—which is entirely possible, although unwanted—causes a reminder to be sent (modal).

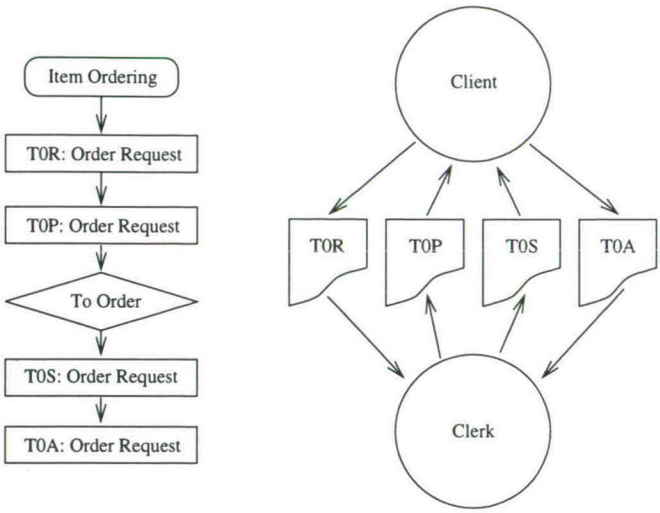

Figure 6.6: The Function and Subject-Communication Models

the counter. Note that the full transaction paradigm as described in Section 6.2.1 might be overkill for the simple activity of requesting an order—it would probably be more efficient to extend the single action TO ORDER with all the actions necessary to get the requested article to the client, after which the factagenic conversation (T0S and T0A) can take place.

In the OI Model, it is irrelevant which subject invokes the action. What is important is the fact that the action type 'to order' applies to both a client and an item, at the same point in time. Neither of the objects fulfills a special role; in particular, it cannot be stated that 'client' is the agent of 'to order' and 'item' is the patient.

Placing action types between object types effectively synchronizes the life of the objects involved. In the KISS Model, which I will not explain in detail, each object type's life cycle is modeled in terms of possible sequences, selections and iterations of actions (in a procedural, deterministic way). If in the OI Model an action is placed between two object types, each object type's KISS Model *must* contain this same action, and in each object's life, this action *must* take place together with the action on its counterpart. If one KISS Model disallows an action to take place at a certain moment of the object's life cycle, the action cannot be invoked at all. In this way, KISS Models govern most of the dynamics and interlocks of a complete domain model.[8]

Newer versions of KISS (Kristen, personal communication) also introduce the notion of an *activity*, thereby emphasizing the instantaneous character of an action ($\delta t = 0$). Activities have specified (instantaneous) but non-overlapping begin and end points in time. Within an activity, other activities and actions can be included. But for the current discussion, the notion of an instantaneous action is sufficient—

[8]The Action Model further specifies restrictions on actions on a lower, more attribute-oriented level.

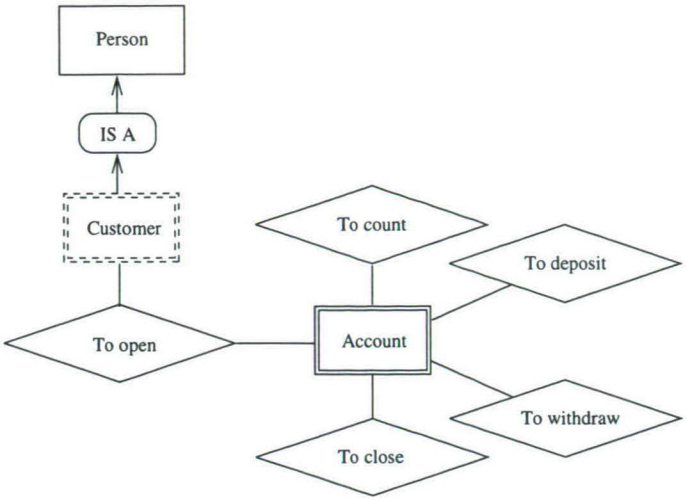

Figure 6.7: Strong and Weak Objects and Specialization

activities always can be broken down into actions, although this would decrease the readability of diagrams.

Most of the objects and verbs in the OI Model could be taken directly from a Lexicon, and the action frames attached to the verbs could be used to scan for other related objects in a textual requirements specification. In this way, a Lexicon could help guiding both human analysts and automated analysis tools by providing common language background.

Weak object types

Some actions cause new objects to be created. If these new objects maintain a strong relationship to their 'parent' object, such as with a bank customer and his/her account(s), it is said that these objects have *dependencies* (Kim and March, 1995).

KISS models objects that are existence-dependent on other objects as so-called *weak objects*, see Figure 6.7. Weak objects cannot exist without their associated strong objects, and the strong objects cannot be removed from the system for as long as there still are associated weak objects. For example, if an account is opened for a bank customer, the account is a weak object and the customer is a strong object: when the customer wants to quit the relationship with the bank, (s)he must first close the account. KISS enforces this 'kill the offspring first'-approach by requiring that the terminating 'close' action type is available only to the *account*. More specifically, only the instantiating action can be shared between a strong and a weak object type. In Figure 6.7, CUSTOMER is also a *specialization* of PERSON, the actual object for which the account is opened. In the next section I will discuss the specialization concept in more detail.

The classifiers 'strong' and 'weak' must be seen in relation to each other; the same object can both be strong (because it has associated weak objects) and weak (because itself is existence-dependent on another object). It can be argued that

any object is weak: the object types in the OI Model that are at the top of the existence hierarchy obviously would cease to exist if the world they live in would be terminated. It is a matter of scope which objects are not considered weak. Since in the KISS method, every object *must* have one and only one instantiating action type, the common solution is that e.g. a bank customer has a 'get to know' action as instantiating action type (because the (legal) person already existed before (s)he specialized into a bank customer). This instantiating action is not shown in Figure 6.7; it would be attached to CUSTOMER and be the first action in its associated KISS Model.

Weak objects can be dependent on several strong objects. For reasons that will be explained in detail in section 6.3, KISS chooses to call weak object types that are dependent on more than one strong object type *gerunds*. In the current context, this only means that neither of the strong objects can be terminated if a single associated weak object still exists. The instantiating action type and the resulting weak object type are often contracted into one single object type that has both action and object features. This process is generally known as *nominalization*. However, OI Models do not need nominalization to increase their expressiveness; it is only a convenient notational shortcut in some cases.

Specializations and Categories

Another major feature of OI Models is called *specialization*, which is closely related to *role playing* of objects. A specialization can be regarded as a new object type from a conceptual point of view, but in the real world it does not actually exist. For example, when a PERSON acquires a library pass, (s)he expands the range of action types (s)he can invoke; but there is no actual instance of LIBRARY MEMBER. In OI Models, however, specializations can be regarded as actual object types. They are connected to their parent object type with an IS-A symbol. An example of a KISS specialization is the role of the CUSTOMER that is being played by a PERSON, see Figure 6.7.

KISS models *metamorphism*, such as a butterfly going through the various phases of its life, as a sequence of specializations of the same object type. To indicate that a certain specialization, such as EGG, must come and be ended before CATERPILLAR, the KISS Models of the specializations are not sufficient; they would contain e.g. TO HATCH and TO PUPATE, indicating the beginning and end of the life of the specialization CATERPILLAR, but say nothing about the order of the respective specializations. However, the KISS Model of the 'parent' object type BUTTERFLY can contain these instantiating actions in the correct order. It is a matter of choice if the actions of the various specializations are or are not completely contained in the KISS Model of the parent object. For reasons of clarity, it is probably best to put the instantiating action types only in the parent's KISS Model. This is consistent with the rule that the only action type that can be shared between strong and weak object types is the instantiating action type of the weak object type.

The last core notion of OI Models is that of a *category*. Whereas specializations are formed by extending the set of possible actions (and attributes) of object types, categories are formed by isolating a subset of shared action types on several object

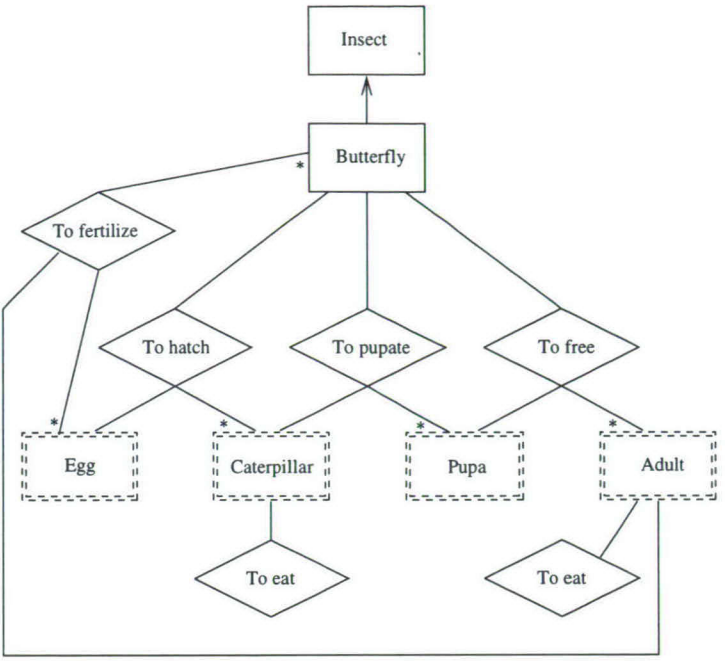

Figure 6.8: A New Object-Interaction Model with Role Playing and Analytical Generalization

types. These shared actions then are detached from the original object types and attached to a newly created, more abstract single object type, with the original object types now *inheriting* these actions from the new abstract type. For example, when both LION and HORSE share the action type TO EAT, it is possible to create an abstract object type ANIMAL, detach the action type TO EAT from both the HORSE and the LION, and re-attach it to ANIMAL.

Specialization such as described above is a *top down* operation, where existing object types acquire and give up features without changing their base identity. Categories are formed *bottom up*, by grouping existing object types together without altering *any* feature of them. For example, a HORSE can be classified as an ANIMAL if the analyst has found that LIONs and HORSEs share some features, and (s)he finds it useful and enlightening to create a new category in order to reference to both types of animal at the same time. Categories such as ANIMAL are a purely abstract invention of a conceptual modeler, used only for his/her own convenience. There are no gains in the creation of a category except for a more transparent model—which is, of course, desirable in itself. Categories also introduce the concept of *polymorphism*, since the observed equal features of the category members are only conceptual and not factual; horses and lions both can eat, but do this in distinctly different ways. When there is no common category to carry the action type TO EAT, the two different versions must e.g. be indexed TO EAT_1 and TO EAT_2.

KISS specializations and KISS categories should be kept clearly separated. Unfor-

tunately KISS uses the same IS-A symbol for the two mechanisms,[9] so that careful observation of the resulting model is necessary. For reasons of clarity and accepted common terminology, I would call KISS specialization *role playing* and KISS categories *analytical generalizations.*

Although both concepts have to do with re-use of actions and attributes, and can easily be mixed up, there is a striking difference between them. This difference does not surface at the syntactic level, but only at the semantic, world knowledge level. It can informally be demonstrated by an example. Suppose there is a world containing two type hierarchies, one stating that 'horses are animals,' the other that 'students are persons.' Both cases seem to be alike, yet there is a crucial difference: *An animal cannot stop being a horse and become a lion, whereas a person can stop being a student and become a teacher.*

I call the horse–animal case *analytical generalization*, because the supertype 'animal' is a convenient, but arbitrary choice—it could also have been 'quadriped'.[10]

The person–student case however is not at all arbitrary, since there is an actual person who is known to have become a student. But the person *does not change* by taking on the role of student; in fact, the same person can play more than one role at the same time. Moreover, the role of 'student' is not truly dedicated to persons only; in a different world, it could be possible that every living creature could become a student, and not only humans. In role playing, there is in fact no actual inheritance involved: the object in question does not change, it only acquires more possible actions.

That in actual implementations it is often convenient to introduce a 'child' object type which carries the new features, leading to object instances with different object identifiers than the 'parent' object, should not distract the analyst from what is conceptually happening in the real world. Implementation aspects, such as the necessity to store grades only for those persons who are students, are completely irrelevant in this respect.

Because it is so easy to mix up various IS-A relationships due to the flexibility of NL, I propose to eliminate the whole IS-A diagram symbol, both for the analytical generalization 'a horse is an animal', and for the role playing variant 'a customer is a person'. Analytical generalization can be expressed by a simple arrow between object types, pointing towards the more abstract supertype—this would be the same symbology that many other OO representation formalisms use. Role playing is initiated by the instantiating action of the role object type in question. The action does not need any special schematic features, since it is a normal action, and the KISS Model of the 'role object' will explicitly put it on top.[11] The role object type can retain its current feature (a double, dashed rectangle, see Figure 6.7). Again, the basic KISS rule that action types are applied to all connected object types *at the same time* makes this an obvious choice: when e.g. an egg hatches,

[9]Linguistically, the two IS-As indeed are expressed in the same way. The NL way of handling specializations and categories blurs the conceptual difference. Extra analytical input is required to separate the two IS-A structures. In section 6.3 I consider ways of doing this. The newest version of KISS explicitly removes the IS-A expression from role-playing specializations.

[10]In natural language, there often is some consensus on what 'typical' supertypes would be, because useful hierarchies have developed in time. However, these hierarchies still are arbitrary.

[11]If the instantiating action type must be visible in the OI Model, it can be marked with e.g. an asterisk * at the meeting point of the object type and the connecting line.

this is the beginning of the butterfly's part of life as a caterpillar—all three object and role types are involved. In Figure 6.8, the resulting KISS diagram is presented. Note that TO FERTILIZE is the instantiating action of both the BUTTERFLY and the EGG. Often it will be quite complicated to find out the proper names for the embracing 'parent' object type—in the example case, I had to use the biological sub-classification for the winged creature that usually is called 'butterfly.' With e.g. chickens and eggs, this problem becomes much harder to solve.

KISS has a terminology problem in the fact that it calls the reverse of specialization 'generalization' and applies this term to the parent of the specialization, e.g. making a PERSON a generalization of LIBRARY MEMBER. However, the term 'generalization' usually is reserved for the abstract classification mechanism that KISS calls 'categories'—a library member is not generalized into a person in hindsight.[12]

Although it might seem intuitively attractive to call the reverse of specialization 'generalization,' both are fundamentally different ways of conceptualizing a world. Furthermore, KISS has a concept 'class' which is neither a category nor a generalization, and also not a 'parent' concept out of which 'instances' can be created, but a group of object instances which share some attribute value—an extensional instead of an intensional concept, highly different from the usual concept called a 'class.'

Complex Objects

Besides the analytical generalization 'is a' and the role-initiating action type, there is another concept that deserves its place in the KISS ranks of conceptual world ordering: the meronymic relationship. Meronymy, or the part-whole relationship, is one of the key concepts in most current conceptual modeling methods. Semantic relationships are often divided into five categories, based on cognitive psychology (Landis et al., 1987):

Antonyms to confront two opposite meanings;

Synonyms to group conforming meanings together;

Class inclusion the well-known analytical generalization or true 'is a';

Part-whole or 'part of'; and

Case relationships such as the agent-patient distinction made on object types associated with an action type.

It is of course no coincidence that all these semantic relationships (unfortunately with the exception of the case relationships) are included in psychologically inspired Lexicons such as WordNet (Section A.2, see also Miller (1993)). In the context of high-level KISS conceptual modeling, the antonyms and synonyms are not relevant, but the other three semantic relationships are quite important. In Section 6.4 I will discuss the case relationships, and in a previous section I discussed the class inclusion. Meronymic relationships are the subject of the current section.

As presented in an comprehensive way by Storey (1991), meronymy can be divided into seven different relationships:

[12] And when it is, it is a case of true analytical generalization, which should be treated as such.

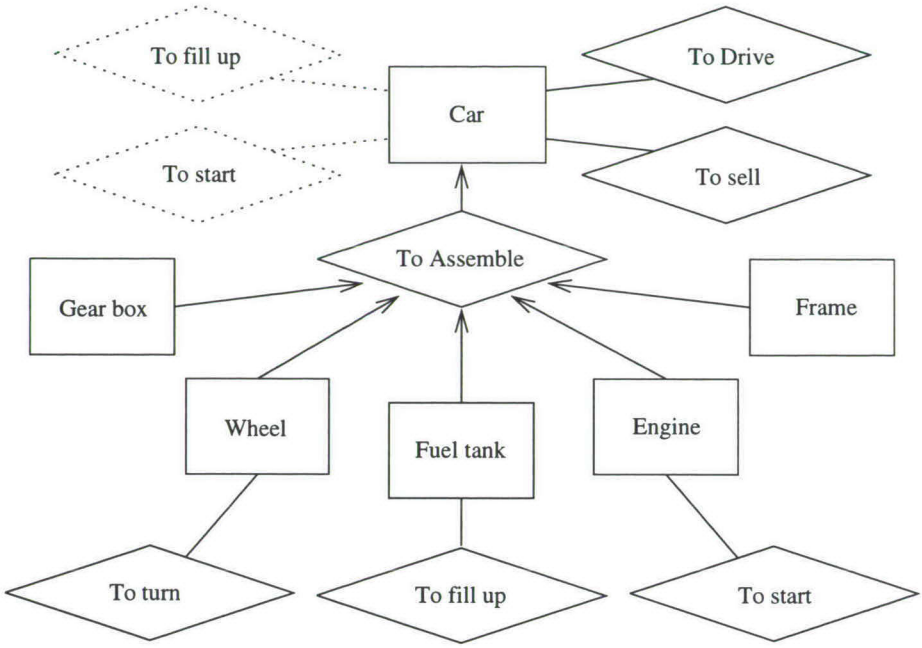

Figure 6.9: An OI Model with Meronymic Relationships and Action Distribution

1. Component–object: 'engine part-of car'
2. Member–collection: 'employee part-of committee'
3. Portion–mass: 'module part-of program'[13]
4. Stuff–object: 'wood part-of desk'[14]
5. Phase–activity: 'billing part-of consulting'
6. Place–area: 'reception-area part-of office'
7. Feature–event: 'demonstration part-of presentation'

The difficulty in incorporating meronymic relationships into KISS lies the fact that not all of the above relationships have a definite starting point in time. Fortunately, KISS itself solves this problem by requiring that each object in the system must be explicitly instantiated. Even when e.g. the reception area and the office are built together, the starting point of the place–area relationship is clear: as soon as both come into existence, the meronymic relationship begins. Therefore, it is only necessary to configure the corresponding actions in such a way that they are all executed at the same time. This is completely analogous to the role-playing mechanism, and suggests a highly similar notation.

However, actions applied to the 'collection' object type may need to be distributed over the participating 'member' object types—instead of action inheritance,

[13]Masses are usually associated to non-set ensembles, such as *water* (Bunt, 1981), but program listings can be seen as homogeneous collections of lines.

[14]Reversed, a desk consists of wood (and iron and plastic).

this is action distribution. The difference can be indicated by drawing the lines between the actions forming the collection and all the participating object types (both member and collection types) with an arrowhead, as showed in Figure 6.9. These arrows cannot be mistaken for analytical generalizations, because the latter are only possible between object types, not between object and action types. In the example, I plotted some actions that are particular to member objects and some that are particular to the car as a collection (e.g., TO DRIVE—you cannot drive a fuel tank). However, some actions I have drafted twice to indicate that they can be applied both to the collection and (some of) the parts, e.g. TO FILL UP. In practice, the actions directly connected to the collection object type should be viewed as *copies* of the 'originals' attached to the member object types—so I drafted them in dotted lines. It depends on the particular view or scope of the diagram which action types are drafted.

When the action TO FILL UP is applied to a CAR, this effectively means that all parts of the car will be sent the same message. Only those parts that can respond to the particular message will actually receive it. For more closely linked objects, such as four legs, a back, and a seat, which are assembled into a chair, TO SIT (ON) will be connected to the collection only, but TO MOVE is distributed.

All of the meronymic relationships such as described by Storey (1991) are covered by this simple notation and distribution semantics. The only possible exception is the *phase–activity* distinction, which might have a better primitive in the standard KISS action–activity pair.[15]

Meronymy could possibly be viewed as normal (multiple) inheritance, with e.g. the car inheriting all properties of the gear box, the wheels, the fuel tank, the engine, and the frame, plus the specific actions that can only be applied to the collection as a whole. But in many applications, this level of detail would be counterproductive; I would need negative or excluding action types to *remove* actions from the collection. The current proposal, possibly augmented with copied action types such as in Figure 6.9, offers enough syntactic tools to model a domain at the necessary level of abstraction and with the correct scope for the application at hand.

Meronymy also could be viewed as role playing, with e.g. the engine playing the role of 'car-engine' for a while. But this would need a mechanism to group all these individual roles together in an abstract 'role' (the car itself), which would gain nothing in comparison with the proposed part-of mechanism.

It is a matter of circumstance whether a car is only considered a car when 'complete,' or whether a car still is a car when e.g. the engine is missing. The last view is in line with cognitive psychological research; it is very difficult to identify the exact property which separates a car from a stack of parts. Glass and Holyoak (1986, p.154) present some examples of gradual property changes and the effect of them on typical object classification. Central in this research is the notion of *prototypes*, objects representing *typical* things. It depends on the domain if a car is considered a car whether or not it contains an engine. Dividing the TO ASSEMBLE action into partial actions, and changing TO ASSEMBLE into an *activity*, can solve this problem.

[15]But for example a model of an organization bureau for festivities might explicitly need to see this as a true part-of relationship.

Table 6.1: Sentence elements and their KISS interpretations.

Syntax Element	kiss Interpretation
Subject	Initiates and controls a specific action
Predicate	The action taking place
Direct Object	The object upon which the action is carried out
Preposition	Additional information in a generic sense
Attributive Adjunct	Distinction between owner/possessor and possession
Indirect Object	Additional objects affected by the action
Adjectives	Attribute of the associated object
Adverbs	Attribute of the associated action or adjective

Care needs to be taken when a car is defined as a car if and only if all its components are assembled, because in this case it is tempting so see the car as a weak object type. Because this weak object type then would be dependent on more than one strong object, it would be called a *gerund*, which clearly is not correct.

6.3 KISS and natural language

Natural language (NL) is deeply embedded in the KISS method, because one of the main design goals for KISS is "...that information systems must always be understandable for the end user and that information systems must communicate with their end users in their spoken and written languages" (Kristen, 1994, p.84). In practice, the NL part of KISS is restricted to the *design* of the information system: natural language concepts guide the information analysis and some parts of the so-called 'information architecture.' What remains of the user-provided NL in the information system does not surpass the usual table headers, field names and user interface labels.

KISS makes a somewhat naive assumption in stating that language can be divided in *vocabulary* and *grammar*, where the vocabulary represents "all the ideas for objects and phenomena that can be observed in reality" and "the grammar [...] includes all the rules and conventions used to create a significant meaning, also known as semantics." This transition between syntax and semantics as assumed by KISS is far too simple, which leads to a way of grammatical analysis that stresses *sentence or phrase structure* instead of 'true' semantics.

6.3.1 Syntactic Analysis

Standard KISS sentence analysis recognizes the base elements which are summarized in Table 6.1. It is clear that these simple rules not always hold (e.g. in a passive sentence, the subject and direct object change positions), but they do provide a way to extract relevant conceptual structures from NL text. If the textual input is pre-structured and written in restricted language (only active sentences in present

tense, etc.), these rules turn out to be remarkably effective.

However, it is not immediately clear if the effectiveness of these rules is truly caused by their underlying syntactic definitions. To my knowledge there has been no sound research on the *actual* way in which KISS analysts find actions and object types in texts. I expect that syntactic analysis will be a good starting point, but that the analysts also add a significant amount of interpretation and background knowledge to their analyses.

The list in Table 6.1 contains both categorical and functional syntax elements. KISS also includes word forms such as gerunds in the set of rules, next to the handling of clauses, question marks, and the imperative. Combined with the assumed background knowledge used by KISS analysts, this heterogeneous mix of linguistic concepts makes me see the KISS way of grammatical analysis as a set of *ad hoc* heuristics instead of as a clean, deterministic process. This view is strengthened because Kristen describes the first step in the KISS analysis as 'the writing of structured sentences' (Kristen, 1994, p.106), taking a leap over about every syntactic analysis and going straight to the semantic KISS analysis. In this way, KISS still depends on the experience and insight of the analyst, using language analysis at the syntactic level only as some kind of *ad hoc* heuristic.

6.3.2 Semantic Analysis

What KISS really pursues is *semantic analysis*, at the conceptual level instead of at the surface level. It is understandable that KISS tries to achieve this goal by referring to the known traditional syntactic concepts, because when the analysis input is plain text, (morpho) syntax is the only readily available description device. Moreover, syntactic analysis can be done by KISS conceptual modelers after relatively little training, building on secondary-school grammar concepts, while semantic analysis with the contemporary state-of-the-art formalisms is not a trivial undertaking. But the relative autonomy of syntax makes the direct mapping of syntax to semantics a hazardous process, and the disappointing results in the area of NLP suggest that the current state-of-the-art in linguistics is not yet up to such a mapping.

The *Grammalizer* project (Hoppenbrouwers et al., 1996; Hoppenbrouwers et al., 1997a)[16] tries to circumvent these problems by using a subset of a semantic formalism that is specially tuned towards the KISS analysis. Taking Dik's Functional Grammar (Dik, 1989) as a starting point, the Grammalizer team defined a restricted set of semantic roles that mapped directly onto KISS concepts and built a tool to assist KISS analysts in directly finding these semantic roles in plain texts. Not coincidentally, the storage structure of the Lexicon as it is defined in this thesis builds on the same fundamental notions. They are shortly summarized in Table 6.2 and will be explained in detail in Section 6.4. Note that some of the tags seem somewhat isolated; this is because more tags were envisioned but not yet implemented in version 1 of the Grammalizer tool kit.

The semantic roles can be attached to words and phrases, and it is possible to recombine them later into KISS structured sentences because the mapping from (restricted) semantics to standard KISS structured sentence syntax is straightforward.

[16]Managed by KISS BV, Erp, The Netherlands and sponsored by the Dutch Ministry of Economic Affairs under Senter grant nr. DIB508313.QID.

Table 6.2: Grammalizer semantic roles.

Semantic Role	kiss Interpretation
Action	The action taking place
Agent	Initiates and controls a specific action
Patient	The object upon which the action is carried out
Other	Additional object affected by the action
Term	A possession or possessed object
Possessor	Something which possesses a Term
Predicate	Predicates something over an Argument
Argument	Something to predicate over
Copula	Filler to complete the predicate structure

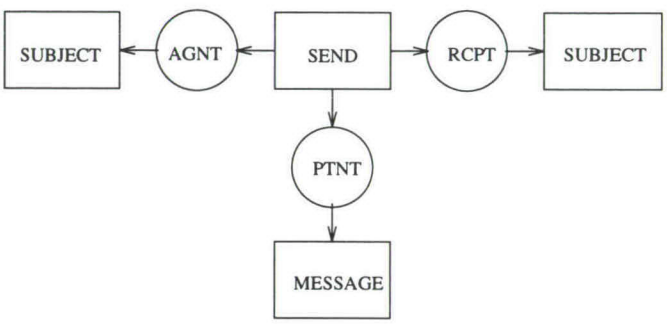

Figure 6.10: A KISS SC Model conceptual graph

In a way, the stacking order of syntax and semantics now is reversed: the syntax is formed *after* the semantics have been determined. This does not solve the problem of bad analysis, but circumvents the false promises that syntactic analysis makes. By skipping the syntactic analysis and directly going to the semantic analysis, the Grammalizer offers a workable frame work for a KISS (upper) CASE tool.

Early experiences revealed that the introduction of linguistic terminology such as in Table 6.2 caused problems with KISS analysts (Hoppenbrouwers et al., 1997a). A later release of the Grammalizer therefore reversed back to official KISS terminology while retaining the semantic meaning of the roles.

For a more extensive discussion of the Grammalizer Project, see Appendix B.

6.4 KISS and Conceptual Graphs

The most linguistically relevant models of the KISS method, the SC and OI Models, have a straightforward mapping to basic conceptual graphs.

The SC Model in the current KISS method only conveys the transfer of a message from one KISS subject to another, which can be modeled as shown in Figure 6.10. As already explained in Section 6.2.1, it would be beneficial for the SC Model to

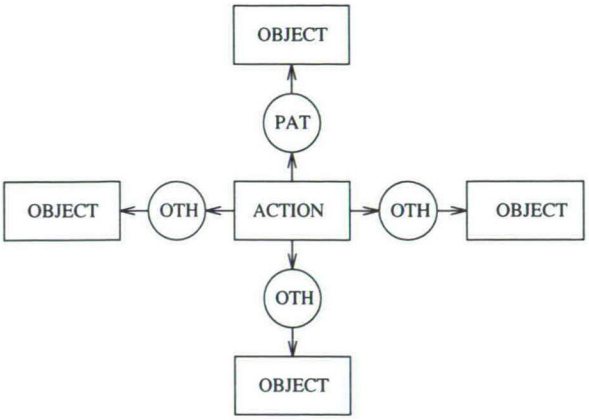

Figure 6.11: A KISS OI Model conceptual graph

introduce several related versions of this conceptual graph. By only replacing the verb SEND with e.g. REQUEST, PROMISE, STATE, and ACCEPT, the SC Model can easily carry the basic speech acts such as used by Dietz' DEMO method (Dietz, 1990).

The OI Model carries far more semantics than the SC Model. I will first present a simple generic case and then extend the CG to include more detailed features. Just as with the CGs for NIAM, there is a complication in finding the appropriate roles for the various conceptual links. With a slight simplification however, I can settle on only two major roles for the generic case (see Figure 6.11). Note that the SUBJECT of an ACTION (i.e., the initiating object) has no representation in the OI Model (it belongs in the SC Model),[17] and the obvious AGENT role is therefore missing.

The number of OTHER roles can vary between zero and infinite, but in practice the number seems limited to at most three. It is possible to subdivide the OTHER role into e.g. LOCATION, INSTRUMENT, BENEFICIARY etc., but deciding which role to use is so tightly coupled to the verb used for the action that the roles should come *with* the action (they are retrieved from the Lexicon)—it is not the analyst's task to provide them. Furthermore, the KISS method does not recognize any semantic difference between the actual roles (with the exception of AGENT), so the only real usage of separate role names would be in their identifying property. Naturally, multiple use of OTHER will not help a great deal in identifying the various roles. KISS traditionally identifies the objects involved in the OTHER roles by position and uses prepositions as semantic cues.

Weak object types, of which the instances are existence-dependent on instances of their associated strong object type, cannot be indicated directly in a CG. This is not amazing since NL does not have the weak object type concept either. Concepts such as existence dependency and inheritance are closely related to analytical, artificial 'views' on a world, and NL has no structures that are especially developed for this 'view' (unlike artificial languages). This is not to say that the weak object concept is unnatural—on the contrary. But it is deeply buried in the cognitive model that

[17]Subjects can appear in the OI Model as well, but then only 'incognito' as passive objects on which the action(s) are applied.

people make of their surrounding environment, and not made explicit in day-to-day communications.

Nominalizations, treated specifically by KISS, have no specific vehicle in CGs; they are represented as straightforward nouns. By separating the action (verb) from the result (noun), and connecting the two with a new verb and a role, this can be solved (remember that nominalizations are not a necessary part of KISS OI Models, they are just handy shortcuts in the notation). Gerunds in KISS are actions with more than one involved object *and* a nominalization, and thus can be represented in a straightforward way as well.

The organization of KISS object types in hierarchies needs two types of support, one for *specializations/role playing* and the other for *categories/analytical generalization.* There is a third hierarchy in CM, which has nothing to do with the supertype-subtype distinctions of the former two, but rather with class-instance relations, e.g., *John is a student.* Since in CM there is no provision for world populations, I am currently less interested in this third variant.

NL uses the same device ('is a') for all three cases of inheritance. Although this is unfortunate, because it easily causes confusion about what form of inheritance actually is used, it also gives NL users a powerful and intuitive way of taking a meta position 'above' the world. Context and conceptual relationships, together with the world knowledge of the NL user, will provide the necessary elements that are relevant to the discourse—there is no *explicit* inheritance involved.

Despite NL's unification of the various forms of inheritance, Conceptual Graphs have a notion of subtypes and supertypes in the *type hierarchy* for concepts (Sowa, 1983, p.80), complemented by the *approximation hierarchy* for percepts, which I will not discuss in the KISS context. Sowa divides the type hierarchy into two cases, *natural types* and *role types*:

> "... the types CAT, DOG, MAMMAL and ANIMAL are *natural types* that relate to the essence of the entities, but types like PET, PEDESTRIAN, and SPOUSE are *role types* that depend on an accidental relationship to some other entity. [...] Natural types and role types both occur in the same type lattice."
> (Sowa, 1983, p.82)

I am not convinced that Sowa's definition of role types indeed covers all roles, such as the aforementioned *student* role. Roles can also be played without an (accidental) relation with another entity, although there usually will be some relationship—in this case, with a university. But the distinction between natural and role types, which also nicely coincides with the psychological evidence of natural categories (see Section 3.4.1), seems appropriate. His suggestion that both categories occur in the same type lattice might need some refinement. Since natural and role types do mix in NL but are conceptually different, and since I argued above that both types of 'inheritance' will be used in different circumstances, I propose to separate the lattices. This is in accordance with the multiple type hierarchy support of the LMS. Also, it enables concepts to participate in more than one hierarchy, possibly in both a natural and a role-based one. Each hierarchy has its own top element and can be distinguished from the other parallel hierarchies.

6.5 Paraphrasing KISS

KISS paraphrasing is somewhat peculiar because KISS separates the SC and OI Models. Analytically, this is a very sound approach which leads to good insight in the fundamental information architecture that underlies the domain model and separates the organizational aspects from the real-world aspects. But the exclusion of the initiating subjects out of the OI Model directly causes a problem when I try to paraphrase the OI 'structured sentences': in paraphrasing, the KISS subjects equate the NL subjects, and the resulting sentences therefore lack the subjects. As an example, Figure 6.5 on page 116 should be paraphrased as 'The ordering of an item for a client' or 'To order an item for a client', because the instigator by definition of the OI Model neither can be the item nor the client.[18]

What is assumed by this particular OI Model is that *a warehouse assistant* orders the item *for* the client, and the corresponding SC Model would include the additional sentence 'Clients send order forms to warehouse assistants': the reception of an order form signals the warehouse assistant to file an order. But even a complete set of models according to the original KISS method does not include any direct link between the warehouse assistant (in the SC Model) and the ordering of an item for a client (in the OI Model).

The extension to Function Models such as drafted in Figure 6.4 *does* allow for such a direct link. Although Function Models are too procedural to be a good candidate for paraphrasing themselves, the extra information they provide can be used to connect subjects to actions.

A complete NL sentence bridging SC and OI Model, thus including a subject, would be: 'Warehouse assistants order items for clients,' possibly with other number indicators instead of the plural. Such a complete sentence can typically be found in user-provided text. The Grammalizer prototype recognizes AGENT (subject), PATIENT (direct object) and OTHER (indirect object) roles, and thus can extract all elements necessary for this type of sentences from a given text.

This results in a situation where the necessary information to correctly and satisfyingly paraphrase an OI Model is available in the start text, and possible in the CASE system, but cannot readily be retrieved because it is scattered in too specialized (sub) models. A good interface could overcome these scattering problems and present the user with clear, concise NL sentences that capture the essence of both OI and SC Models at the same time by means of the revised Function Model.

6.6 KISS and the Lexicon

One of the design principles of the KISS method is that each used label must be unique, unless there are multiple occurrences of the same concept. For example, it is not allowed to have both 'A bank opens an office' and 'A client opens an account', because the word *to open* refers to a different action type. When an appropriate replacement word is missing, KISS usually postfixes a number ('open_1') to distinguish the concepts.

[18] Actually it *could* be the client, but this cannot be indicated in the OI Model.

The plain fact that NL *does* use the same word twice suggests that there at least is some common ground between the two concepts, and KISS's way of separating them at the lexical level therefore is rather crude. The LMS identifies concepts and frames not by means of their appearance (their lexical or string form), but solely by their usage. In this case, one of the frames with the lexical 'open' takes a bank organization as agent and an office as patient, while the other frame takes a customer and an account, respectively. Further identification is possible because concepts also belong to a domain and have a particular place in the type hierarchies. Lastly, each stored concept has its own object identifier which is assigned by the LMS.

The LMS both solves KISS's identification problem and offers support for the NL 'common ground' by separating lexicals and concepts. KISS action types and object types, both in the SC and OI Models, map to LMS frames and concepts. The KISS labels equate lexicals.[19] Each LMS concept has a set of lexicals, at least one for each supported language. A complete set of KISS models would contain no actual strings as labels, but only references to the appropriate concepts in the LMS. The LMS then would provide the correct (forms of the) lexicals on request.

What the LMS then offers is *a complete separation of the conceptual model and the language it can be expressed in*, opening not only the door to easy translation of conceptual models, but also to re-paraphrase the same model in totally different and potentially clearer ways. As an example, KISS often models the result of an action as a gerund, e.g. an action 'to insure' leads to an 'insurance.' The LMS now offers the possibility to combine both the action and its result into the same concept. The frame which connects a client and the object which must be insured (the actual action 'to insure') gets references to two lexicals: one English verb, 'to insure,' and one English noun, 'insurance.' Depending on the purpose of the paraphrasing routines, either one of these lexicals might be used to form an appropriate sentence. In this way, the actual CM gets more transparent because an action and its result now not only share a common NL root form, but are actually a unified concept.

[19] Actually, the labels equate specific *forms* of lexicals.

Chapter 7

Conclusions

— in which the author briefly summarizes the results presented in this thesis, discusses them in the light of other approaches in the same field, and indicates some directions in which further investigations should be considered —

The previous chapters discussed various topics concerning Conceptual Modeling and the Lexicon, often from different points of view and in the light of several existing modeling paradigms. The current chapter aims to present a coherent overview of the conclusions.

7.1 Conclusions

Requirements Engineering is a partially informal process that inevitably involves the use of natural language in many process phases (Section 2.1). Various methods exist to handle NL specifications more or less formally in order to produce formal models. However, approaches that consider NL—in particular terminology—to be an integral part of the method are rare. Disregarding the knowledge available through terminology unnecessarily complicates the analysis and the subsequent verification and validation phases. Centralizing the domain terminology and standardizing on a common representation for words and a common meaning for concepts, better structured than a traditional Data Dictionary, would improve this situation (Section 2.2). Such a 'Concept Dictionary,' or Lexicon, deserves a definite place among other CASE tools, such as repositories, because it provides the terminology to connect the modeling paradigm's abstract world view with the users' intuitions.

The field of linguistics is a rich source of lexicographic methods. Research has shown that a proper Lexicon should contain a large amount (tens of thousands) of lemmas, not restricted to the domain in question, organized in a number of ways (Section 3.1). Theoretical frameworks such as Functional Grammar propose generic, flexible frame structures to store lexicographic and semantic information, and suggest particular Lexicon organizations (Section 3.2). Empirical methods can also be used to form hierarchies based on cognitive aspects of words (Section 3.1.2). However, there is no 'best' way to organize the semantic concepts in a Lexicon. On the other hand, morphosyntax follows strict rules and can quite well be organized in a single structure. Strict separation of morphosyntax and semantics therefore

is preferable, as long as the relationships between concepts (*signifiés*) and symbols (*signifiants*) are retained. The resulting structure profits from object-oriented primitives and is expressive enough to cover a wide range of applications.

Key assumption in the Lexicon is that no concept can ever be defined unambiguously by enumeration of its properties. For every concept, the domain in which it was coined and the links with other concepts in the same domain uniquely 'define' it in terms of *usage*. When a Lexicon is used to make the transition from one domain to another, these implicit definitions can be used to learn the meaning of the alien terminology in much the same way as with natural second language acquisition (Section 3.4.1).

Combination of Lexicon structural requirements with database technology led to the design of the Lexicon Management System (Chapter 4), a formally described set of classes which together form a storage system for both morphosyntax and semantics. Some examples were given to illustrate the expressive power of this LMS (Section 4.3). A generic Lexical Information eXchange (LIX) language (Section 4.4) has been developed, that can be used both to specify Lexicon Schemas and to manipulate the Lexicon contents. A Lexicon has been designed and partially implemented, which fulfills all the above requirements and is flexible and scalable enough to be adapted to various industrial applications (Appendices B and C).

NIAM claims to be heavily NL-based, but closer analysis reveals that many of NIAM's NL primitives are mainly simple labels stuck onto relational concepts. It turns out to be difficult to capture NIAM in a generic frame representation because NL frames do not agree well with relational concepts (Section 5.2). Therefore, standard NIAM is not truly fitted for NL analysis. However, with several minor changes to the approach (not the notation), NIAM can be related much better to NL frames (Section 5.4). This would not necessarily improve the conceptual correctness and quality of NIAM analyses, but it would improve the link between conceptual NIAM models and the mental domain model of the domain specialists. This, in turn, might lead to conceptual models that are easier to create, and more importantly, easier to verify and validate. A new way of NIAM modeling could be introduced that first models a domain in terms of NL frames and concepts, trying to stay close to the domain specialist's own observations, followed by a specific 'technical' model which lays out the fundamentals of the information system. The first (domain) model then could be used as an intermediate model by means of which the facts and constraints of the technical model could be explained. Another possibility would be to determine a set of practical transformation rules, not necessarily covering all cases, with which analysts could turn a domain model into an implementable technical model. This would provide a relatively stable business objects layer—the domain model—that would change with the organization, and a technical layer that would change with the information system.

KISS is more closely related to NL, because many KISS primitives have been derived directly from NL concepts (Section 6.2). However, in some critical areas there are still gaps that should be closed before KISS can truly be called NL-based. Especially the link between SC and OI model, both of which naturally map to NL frame representations, is missing, but it can easily be added. This would lead to a substantial improvement in both expressiveness (e.g., taking authority and instigation into consideration) and conceptual elegance. Addition of a Lexicon to

a KISS CASE tool would provide analysts with direct access to most NL frames and roles required for such a link. Possibly, a Lexicon would improve initial analyses as well, by facilitating the parsing of NL requirement documents for typical message and action words. KISS analysis practice is currently loosely based on NL analysis (Section 6.3) but the introduction of semi-automatic NL analysis tools in the form of the *Grammalizer* package (Appendix B) may significantly enhance the link between NL and KISS, and take over some routine labour as well. The Grammalizer has already proven to be powerful enough to capture most of the core KISS concepts in NL-based frames.

7.2 Discussion

The results and conclusions as summarized in the previous section should be viewed as indications that conceptual analysis based on NL concepts offers interesting points of view. I do not claim that the suggested analysis approaches in the NIAM and KISS example cases are optimal; they reveal different relationships between concepts than do the traditional, logic or technology driven methods, but probably cannot replace these methods completely. A coherent combination of NL-based analysis during the first analysis phases with traditional methods during the later phases might prove most effective.

There have been earlier attempts to combine NL and conceptual modeling, such as GRAMMARS (Dijk et al., 1989). This particular method describes database tables with fixed-format sentences, following the frame SUBJECT - VERB - OBJECT - {PREPOSITION - OBJECT}.[1] Because GRAMMARS is heavily biased towards the relational data model, it has many features that also can be found in NIAM, including explicit transformation rules to generate SQL tables. However, GRAMMARS uses a language-oriented approach which resembles the way in which KISS looks at the world, with action frames centered around verbs. Typical for GRAMMARS is that the NL sentences themselves can be stored in relational tables, which can then be used as blueprints for an actual (prototype) information system. GRAMMARS does not exploit the deeper semantics of agent/instigator roles or any other concept; the words are used as mnemonic concept handles only. Therefore, domain terminology is not used explicitly, and GRAMMARS in itself would not profit from a Lexicon.[2]

Because this thesis concentrates on fundamental and theoretical issues, I have provided only limited empirical evidence of the viability of the suggested approach. Earlier work, especially by Weigand, indicated that there was a definite link between CM and NL, but did not provide a structured overview of these links. My thesis tries to point out at least some of these links, and also suggests actual approaches and tools to exploit them. Preliminary experiences with the Grammalizer as a NL frame-based KISS analysis support tool indicate that the main links between CM and NL concepts seem to be sound.

From the start, it was clear that my PhD project had to drive towards actual implementation of a workable Lexicon, and the current developments in both the Grammalizer and the TREVI projects indicate that this goal has been reached. How-

[1] There are in total four different sentence frames in GRAMMARS.

[2] Unless when the Lexicon is used to normalize word forms from to infinitives and singulars.

ever, there is a lot more that needs to be done before NL-based analysis finds its way into mainstream information technology, both in working practice and in industrial-strength tool kits.

7.3 Future Work

Obviously, the Grammalizer and TREVI projects, which are both still underway as I write this, will turn out interesting results that should be carefully examined and might lead to new insights, both in theoretical and in practical Lexicon applications.

What still is needed as far as CM is concerned is a concise method to acquire all the lexical information discussed in this thesis *in a standard manner.* Many current CM methods focus on representation instead of on acquisition, and it is commonplace that several equally experienced analysts produce different analyses of the same domain. This might be because the current CM methods still incorporate a significant design aspect, where personal contributions of the analyst can and will slip in. Any CM method should strive to eliminate design decisions as much as possible, including hierarchical tree structures if they cannot be unambiguously calculated or discovered through empirical research. A 'top' domain model based on an NL Lexicon might be more consistent between analysts, because they would not need to invent all concepts from scratch and could connect to a large existing base of common frames and concepts. However, this is an hypothesis only which should be verified by empirical research.

Other possibilities for Lexicon applications to IT can be found in the areas of distributed systems, where unrelated services need to communicate (possibly through intermediate brokers). If these services have no previous knowledge of each other's features, they can use a Lexicon to match their terminology against a common base of concepts. On a smaller scale, even single systems quickly grow towards an architecture where so many unrelated objects (from different programmers or even vendors) have to exchange messages that it becomes impossible to match all their interfaces by hand. A Lexicon which contains the domain terminology for the system may help the objects in locating their peers and communicating with them. All these kinds of distributed systems will not be fully deterministic, because a Lexicon does not guarantee that objects correctly understand each other, but their sheer complexity might require such fuzzy communication methods.

Lastly, the increasing volume of electronic data interchange (EDI) and electronic commerce offers a unique challenge to the many systems involved, because it becomes impossible to standardize on all required aspects of commercial data traffic. Development of autonomous agents which can negotiate over contracts might benefit from Lexicons with common terminology, offering an escape route from current standardization problems.

Part III

Appendices

Appendix A

Existing Lexicons

Many, if not all, linguistic theories have at one point described the Lexicon and embedded it among the other features of its particular framework. But as observed in Section 3.1.1, preciously few full-blown implementations actually came to life, probably because the development and proof of the theory had precedence over the collection of large amounts of, in a sense, superfluous data—one word of every recognized category was sufficient to make the implementation work.

Some projects, however, started with another aim. They intended to produce a fully fledged computational lexicon, not bound by a particular linguistic theory. The already described Acquilex project (Copestake et al., 1992) was such a project, albeit influenced by traditional lexicography and the availability and content of machine-readable dictionaries. Cyc (Lenat and Guha, 1990; Lenat, 1995b; Lenat, 1995a) does not aim at a computational lexicon, but at a 'common sense knowledge' base. Cyc is claimed to be a universal schema consisting of roughly 100,000 general concepts, spanning all aspects of human reality. This ambitious goal might also be its major problem, because as I argued above it is inherently difficult to provide 'all things for all people' and still maintain a satisfying level of practical usefulness. Because Cyc is not readily available for the purpose I need, I will not expand further on its structure and contents. The same holds for the Japanese Electronic Dictionary (EDR) Project (EDR, 1988; Yokoi, 1995b; Yokoi, 1995a), of which results are commercially available but therefore of less interest to research.

Two projects of which the results are freely available are CELEX and WordNet, which I will briefly discuss in the next sections.

A.1 The Celex Lexical Database

The CELEX Centre for Lexical Information (a joint enterprise of various Dutch university institutions), founded in 1986, has compiled three large electronic databases which can provide on-line and off-line users with detailed English, German and Dutch lexical data. The Dutch database, version N3.1, was released in March 1990 and contains information on 381,292 present-day Dutch word forms, corresponding to 124,136 lemmata. The latest release of the English database (E2.5), completed in June 1993, contains 52,446 lemmata representing 160,594 word forms. The German database (D2.5), made accessible in February 1995, currently holds 51,728 lemmata with 365,530 corresponding word forms.

Apart from orthographic features, the CELEX database comprises representations of the phonological, morphological, syntactic and frequency properties of lemmata. Furthermore, information is being collected on syntactic and semantic subcategorizations for Dutch.

A.1.1 Contents and Organization

For all three languages, CELEX provides three so-called 'lexicon types': lemmas, word forms, and corpus. The *lemma lexicon* contains a list of head words (cf. traditional dictionary entries). The *word form lexicon* yields all possible inflected words: every entry in this lexicon is an inflectional variant of the related head word or stem. The *corpus lexicon* provides an ordered list of all alphanumeric strings found in the corpus with raw string counts, not disambiguated for relations to either lemmas or word forms.

Having pinpointed a lemma, a CELEX user may select a large amount of additional information, combining information on the orthography, phonology, morphology, syntax, and frequency of the entries. Most information is fairly well detailed and appropriate decoding programs can convert it into almost any required representation.

Currently, there is no higher-level syntactic[1] or semantic information available; CELEX is a purely lexical database. As such, its usages are limited to lexical research into e.g. word frequencies, to retrieve lemma sets for given word forms (CELEX offers no support in disambiguating lemmas), and to produce the appropriate word forms given a lemma and a morphosyntactic specifier. It has also been used for certain psycholinguistic experiments.

A.1.2 Implementation

All lexical data are contained in an Oracle RDBMS, allowing users to make individual selections from the vast quantities of data included. The CELEX user interface FLEX has been especially designed to facilitate access to, and use of the databases. By means of the menu-driven FLEX user interface, queries can be executed on-line and lexicons can be extracted for off-line applications. A 'flattened' version of the database in the form of ASCII files can be obtained on CD-ROM, including some limited low-level access scripts.

A.2 WordNet

WordNet is a remarkable lexicon in the sense that it is based on psycholinguistic fundamentals . Instead of choosing word category, syntactic behavior, or orthography as the way of ordering, it groups the words together by means of *synonym sets*. Beginning with word association studies at the turn of the century, psycholinguists have discovered many properties of the mental lexicon that can be exploited in lexicography. The WordNet project, started in 1985 at Princeton University, started out

[1] CELEX indicates the word type, such as *verb* or *noun*, but not the possible restrictions on usage.

to provide an aid to search in a lexicon in a conceptual rather than an alphabetical way (Miller et al., 1993).

Effectively, this implies that WordNet is organized in terms of word *meanings* rather than word *forms*, and thus resembles a thesaurus more than a dictionary. A Lexicon can indeed can used both ways (Calzolari, 1988). The basic design of WordNet separates word forms and word meanings, and the starting point for WordNet's lexical semantics is the mapping between these word forms and meanings. When the same word form connects to two or more meanings, there is polysemy; when the same word meaning connects to two or more word forms, there is synonymy (relative to a context). WordNet considers word meanings to be synonymous "... if the substitution of one for the other in [a linguistic context] C does not alter the truth value" (Miller et al., 1993). The notion of linguistic context is essential. This definition of synonymy in terms of lexical and syntactic substitutability makes it necessary to partition WordNet into nouns, verbs, adjectives, and adverbs.[2]

WordNet recognizes more relationships besides synonym sets. There are antonym sets (between word *forms* instead of meanings), hyponymy/hypernymy relationships, meronymy/holonymy relationships, and others (Miller et al., 1993). The way WordNet handles morphological relations (so that it can trace 'trees' when the database only contains 'tree') has been built outside the core database—it is a separate program contained in the WordNet interface. The whole way in which WordNet defines word meanings is based on the various relations between (possibly singleton) sets.

A.2.1 Nouns in WordNet

Nouns in WordNet are typically stored as a part of (usually) a single inheritance hierarchy by explicit hypernymy relations, added by the lexicographer. The hypernymy tree of its superordinates therefore can be used to 'inherit' certain knowledge (such as relations to other words, e.g. meronymic links) about the noun in question, saving space and facilitating maintenance. Since every noun is included in a synonymy set (synset for short), the inheritance hierarchy in fact connects synsets. By looking at the superordinate synsets, a WordNet user can compare the noun in question to other, presumably partially known nouns, and in this way construct some mental image of the noun. To help with this process, WordNet contains many true dictionary-like NL definitions as well, called glosses.

In order to avoid all the nouns together to be part of a single inheritance hierarchy with very generic top elements (such as 'entity'), WordNet uses 25 unique beginners for noun hierarchies. In choosing these 25 main categories, total coverage of all known nouns was considered most important, and therefore there is some overlap. Next to the 25 main categories, the WordNet designers used a small group of abstract categories to include some hierarchical relations between some of the 25 main categories (see Section 3.4.1).

Besides the inheritance hierarchy (which points to other nouns), nouns in WordNet can be connected to adjectives (providing attributes), other nouns (which pro-

[2]WordNet also recognizes function words, but considers them to be part of the syntax and not of the lexicon.

vide parts), and verbs (which provide functions or actions the noun[3] can perform). However, many of these relations are not included in the current WordNet database.

A.2.2 Verbs in WordNet

Verbs in WordNet are divided into 15 groups, largely on the basis of semantic criteria. 14 of these groups cover semantic domains such as change, cognition, communication, and competition. The remaining group contains verbs that share no semantic properties other than that they refer to states. WordNet uses elements out of both decompositional and relational semantic analysis and tried to provide the best of both worlds (Fellbaum, 1993). Unlike nouns, there is no straightforward way in which verbs can be ordered in a hierarchy. Fellbaum gives various ways in which the verbs can be arranged into tree structures, but most of the 15 groups require their own particular structure. Verb hierarchies tend to have a much more shallow structure than noun hierarchies, usually up to four levels deep. Many other ways to categorize verbs are discussed in Fellbaum (1993), including opposition relations and causal relations.

A.2.3 Adjectives and Adverbs in WordNet

WordNet also includes adjectives and adverbs.[4] Adjectives are divided into two major classes, descriptive (typically bipolar attributes, such as 'low' and 'high') and relational (something like 'of, relating/pertaining to, or associated with'). Descriptive adjectives can often be graded between their extremes (ancient–old–middle-aged–mature–adolescent–young–infantile), whereas relational ones cannot (criminal [law]). Relational adjectives are usually connected to a noun, which is why the creators of WordNet chose to express much of these adjectives' semantics by including a pointer to the appropriate noun.

A.2.4 Implementation

WordNet was implemented in two separate parts. The contents are provided by lexicographers in plain ASCII, using a technical but convenient notation. To improve manageability, the whole lexical 'source' has been split up in related parts that can be handled as a whole. A compiler called the *Grinder* is used to merge all this lexical information into one (main memory) representation, and then produce an optimized database structure that can be used by machines. This database structure consists of a data file and an index file, structured in such a way that they do not need to be completely pulled into main memory by a WordNet browser.

Besides the Grinder, the WordNet system consists of several browsers, for different platforms and user interaction paradigms. All these browsers work on the same database files so that an update of the lexical contents of WordNet can be done independently of the browser software.

[3] Better: the object the noun represents.

[4] I could not find detailed descriptions and discussions of WordNet's handling of adverbs. In this thesis, I will limit myself to the adjectives.

WordNet only stores 'normalized' word forms, such as infinitives for verbs and singulars for nouns. The browser software contains a morphological module that casts incoming requests for word forms into the normalized database forms. Therefore, it is not trivial to get information about which morphological form (e.g., first person plural) has been input.

A.3 Conclusions

It is striking that whereas CELEX only stores relatively low-level morphosyntactic information, WordNet is restricted to higher-level semantics. Where CELEX excels in morphosyntactic completeness but falls short in any semantic query, WordNet bypasses any morphosyntax and concentrates on inter-word relationships and semantic features. Both lexical databases combined would result in a great wealth of lexical information in the broadest sense, especially when again combined with the results of MRD extraction projects such as Acquilex.

The Lexicon Management System such as described in Chapter 4 would be a good storage system for all this information. Further work to use CELEX, WordNet, and possibly Acquilex data to combine it all into one single Lexicon, governed by a custom-made lexicon engine instead of by a commercial RDBMS (which is not tuned to the specific structures a Lexicon requires, and therefore complicates any query), may well turn out to be very productive (see Appendix C).

The recently started EuroWordNet project seeks to develop a *multi-lingual* lexical database with word nets for several European languages (Dutch, Italian, and Spanish). The European word nets will, as much as possible, be built from available existing resources, and CELEX might play a role as a supplier of raw material. Furthermore, the developers of EuroWordNet will try to merge the major concepts and words in the individual word nets to form a common language-independent ontology.

Appendix B

The Grammalizer Project

The *Grammalizer* project (Hoppenbrouwers et al., 1996; Hoppenbrouwers et al., 1997a) is a cooperation between KISS BV (Erp, The Netherlands), Tilburg University/EIT, and Amsterdam Free University, supported by the Dutch Ministry of Economic Affairs under Senter grant nr. DIB508313.QID. The project participants work together on the development of an industrial CASE tool, based on the KISS Method for object orientation and natural language analysis.

B.1 Basic Grammalizer Features

The KISS Method does not only stress the importance of NL concepts in the various modeling primitives and paraphrasing of models. A paramount phase in any KISS analysis is the initial knowledge elicitation from domain experts using *formal textual analysis*. Although many conceptual modeling (CM) methods start with free NL text as input, few of them have a formal framework available to analyze free input text in a coherent way. Knowledge elicitation through textual analysis is closely related to information extraction (Cowie and Lehnert, 1996); both approaches aim at isolating tiny bits of information out of large amounts of text.[1]

When initial requirements for an information system have to be collected, there usually is a large amount of written text available (manuals, organization descriptions, procedural guidelines) which was not explicitly written as a domain description document, but nonetheless contains valuable information. Formal analysis of these documents by an experienced analyst usually reveals many KISS concepts (subjects, messages, objects, actions) that are worth to be introduced in the initial models (Kristen, 1994; Hoppenbrouwers et al., 1996; Hoppenbrouwers et al., 1997b). The Grammalizer tries to take over the burden of manual language analysis, at least partially, and produce initial KISS models from NL input texts.

The Grammalizer team is well aware of the fact that the state of the art in natural language processing is not nearly advanced enough to provide a fully automated text parser at the semantic level (Wilks, 1996). Fortunately, the primitive KISS concepts are closely related to the morphosyntactic language level, so it is possible to take some shortcuts to extract the relevant details. It is expected that neither a full syntactic nor a full semantic parse will be necessary to extract the initial models.

[1] The TREVI Project (Appendix C) is a project which includes true information extraction.

Moreover, help of an experienced KISS analyst will always be required to clean up the initial conceptual models, because most domains are not formalized rigorously enough before any attempt at modeling is made. This means that even if it were feasible do a full semantic parse on an input text, the result would probably still be semantically unacceptable.

This approach led to the design of the Grammalizer, a CASE tool which supports a structured analysis, following fixed guidelines, of unstructured free NL input text. An analyst can use this system as a tool which offers structuring and memorizes the analyst's decisions, augmented by limited capabilities for automatic text analysis. The tool is not in the first place intended for stand-alone NL text parsing without analyst post-processing, but plays an important role in the whole analysis process.

B.1.1 The Grammalizer Process

The Grammalizer process consists of six phases, which can be executed in logical sequence, although in practice there will always be regressions to previous phases (see also Kristen (1994)).

Text pre-processing A rough separation of the available text in relevant and irrelevant material, usually at section or paragraph level. This phase is also meant to divide the input text in manageable chunks of about one page each, so that the resulting initial model is still within the intellectual grasp of both the analyst and the domain specialist.

Text tagging Some KISS concepts are so closely related to morphosyntax that straightforward 'felt tip marker pen' tagging in the plain text is sufficient to isolate and categorize the appropriate words. Other concepts need more elaborate work, e.g., connecting lower-level tags together and classifying them under higher-level 'container tags,' as with ⟨preposition,indirect object⟩ tuples which are part of an action frame.

Templating The textual elements that were tagged in the previous phase are formally put together by filling template slots with the corresponding tags. When there are not sufficient tagged words available to fill up all slots in a template, the analyst has to provide the missing information (usually after consulting the domain specialist). There are many templates available; the Grammalizer makes a preselection out of them based on the available tagged words. The analyst then takes over and completes all missing parts, calling up new templates if necessary.

Initial model building An automated procedure translates the filled templates into an initial model, which is exported to a dedicated KISS drawing package. According to these drawings, a KISS Domino model is laid out on a flat surface. Such a model consists of specially prepared 'domino bricks' and allows easy, interactive remodeling.

Domino session By means of a KISS Domino session with domain specialists, the initial model is refined and verified. If appropriate, various KISS primitives

that have no explicit equivalent in NL are added, and some lower-level details can be put in as well.

Confrontation The resulting Domino model is copied with the KISS drawing package and translated back into filled templates, so that it can be confronted with the original templates. Where there are significant differences, the analyst has to provide an explanation to keep the modeling process traceable. Such a deviation means that the original textual description was not followed, which could have significant implications for some other aspects of the system.

The first three phases are comparable to the typical components of an information extraction system (Cowie and Lehnert, 1996, p.85). The last three are specific to the KISS analysis. Note that all of the usual remodeling due to errors is put in the fifth phase, the KISS Domino session. The resulting model is assumed to be verified and final.

After the confrontation with the original text and the corresponding justification of the changes, the KISS conceptual model is completed by using the KISS drawing package. Various technical details are added and eventually a complete (prototype) system is generated directly from the specifications. The Grammalizer is not involved in these final steps; its usefulness ends with the confrontation of the initial model with the final model.

B.2 Version 1

Version 1 of the Grammalizer, intended to be a prototype to assess the feasibility of the theoretical approach, was implemented in 1996. For reasons of portability and flexibility, the team used the well-known Tool Command Language (Ousterhout, 1994) and distributed the system over several well-defined, small modules. Because the modules communicate over generic TCP/IP socket links, and because Tcl implementations are available on all major platforms, this approach provided us with the required flexible client-server architecture.

The prototype proved the viability of the architecture, as we successfully ran the system on a combination of Sun Solaris and SunOS, Linux, and Microsoft Windows NT workstations. The system underwent several usability revisions without major problems. Although initially not anticipated, the prototype proved to be so usable that it went into limited production. The absence of some functionality such as project and configuration management prevented a full-scale introduction in the KISS organization.

The prototype contained four separate modules with well-defined tasks.

1. The **Marker** was used to manually mark up an input text with linguistic tags, such as *agent* and *predicate*.

2. The **Elementizer** allowed the analyst to manipulate the various frames that the mark-up process produced.

3. The **Typer** normalized the frames into strict KISS format, including word-level rewrites to infinitive form for verbs and singular form for nouns.

4. The **Lexicon** provided the necessary (morphosyntactic) information to do the rewrites.

B.2.1 Experiences with the Prototype

In order to gather user feedback and empirical data on the actual way in which the system was used by KISS analysts, we set up some extensive interviews and think-aloud sessions. A free NL input text was used to provide the required raw material. We took a well-known standard domain for this text, a limited financial bank application, since we were primarily interested in the way people used the Grammalizer, not in their KISS analysis skills. We set up a stand-alone Grammalizer system in a quiet environment and made sure the subjects were not under time pressure.

After a brief explanation of the Grammalizer functionality and the user interfaces of the three modules of the prototype (all graphical and, in our opinion, straightforward), we presented the analyst with the input text (already in electronic form, but without any tags) and started a tape recorder to record the conversation between the researcher/instructor and the analyst. A second researcher took additional notes in the background without interfering. We explicitly instructed and encouraged the analysts to speak aloud about what they were thinking and doing, and to make any comment they wanted. Questions asked to the researcher were answered as concise as possible, obviously keeping actual active guidance to a minimum.

Observations

The think-aloud sessions, together with interviews before and after the sessions, indicated an unintended gap between the originally developed Grammalizer process and the working practice of KISS analysts. Contrary to our expectations and the promoted way of performing a standard KISS analysis, most analysts approached the presented text in a rather unstructured way. They appeared to have a predetermined idea of which word types (nouns, verbs) would result in usable KISS concepts and often only scanned the text superficially. Without the Grammalizer, they would have produced semi-structured lists of 'interesting' words; with the Grammalizer, they often tried to do the same. But because the Grammalizer has a set of strictly hierarchical tag order rules, it is not possible to just tag words at random. Some analysts thought this was an unnecessary restriction to their insight and creativity.

There was a striking performance difference between analysts who were from the start sympathetic towards a textual analysis tool—even if they indicated that in production it might not yet be practical—and those who rejected the tool. Whenever a thorough textual analysis with the Grammalizer was seriously performed, the results were generally correct; with a superficial analysis, the results were unsatisfactory.

Further performance analysis revealed that cooperative subjects showed a higher creativity than expected in using the predefined tags to accomplish analyses. It turned out that most of the creativity was triggered because the analyst 'saw a KISS concept shimmering through the text' and applied the linguistic tags in such a way that the net result produced the intended KISS analysis, even if the linguistic text analysis was now incorrect. The Grammalizer was not built to block these creative

tag applications.

A last observation was that the linguistic nomenclature of the tag set (e.g. *agent, patient, predicate, term,* and *possessor*) was perceived to be difficult and confusing, although the basic analysis performed with these tags was done correctly.

Implications

After the tests had ended, we abandoned the linguistic tag labels and replaced it by KISS terminology *without actually changing the application of the tags.* Our hypothesis was that this renaming would not decrease the analytic performance of motivated analysts, who already showed to be skilled in creative tag application, but would lower the threshold for less motivated analysts because they now could keep thinking in the familiar KISS frame work. The KISS method is already heavily based on low-level morphosyntactic concepts, and the loss of linguistically correct nomenclature for these concepts seemed a small sacrifice if we could gain more acceptance of the tool.

Another finding from the interviews and think-aloud sessions was that many analysts preferred to only superficially scan the input text for basic domain concepts, and wanted to immediately start modeling with KISS Domino. This rush approach to modeling is not ideal, but explicitly disallowing it seemed not attractive either, because then we would encounter serious motivation problems. Therefore we decided to offer more support for direct sentence authoring under template control by de-emphasizing the tagging process (partially through automation) and focusing more on the template engine.

B.3 Version 2

Because the second version of the Grammalizer was intended to be used in large-scale projects and also should prove itself as a commercial tool, we divided the tasks at hand into various functionally isolated subtasks, and assigned each subtask to a stand-alone module which was optimized for the job (Figure B.1). It was decided that we would not produce a monolithic system, although for the various users it had to appear as an integrated whole.

From a user's point of view, the Grammalizer V2 consists of two modules, the *Tagger* (a tag engine) and the *Analyzer* (a template engine). Both modules have a graphical user interface and are coordinated, in the sense that relevant changes in one module are carried over to the other module on-line. A user can only start the Tagger from within the Analyzer.

A functional view of the Grammalizer system reveals more dedicated modules. Besides the two graphical user interface modules, there is a *Request Broker*, inspired by the CORBA specifications (OMG, 1996), which channels all inter-module communication through one well-monitored gate. This Broker has a real-time overview of which interface modules are active (including multiple usage of modules and models by multiple users) and coordinates the requests, e.g., it prevents a model to be opened for editing by more than one user. The Broker can call in services of technical modules such as a Project Repository (for persistent storage), a Log Server, a Lexicon Server, and various algorithmic automation units such as the AutoTagger.

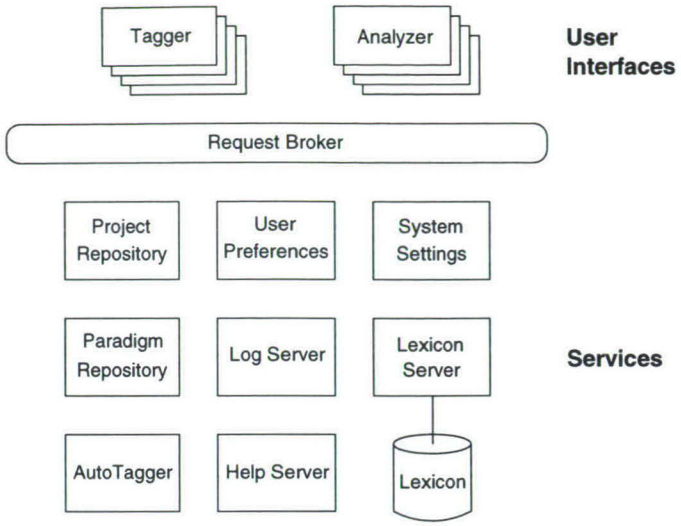

Figure B.1: The Grammalizer Architecture

The AutoTagger itself can call in further assistance from specialized modules, e.g., a rule-based NL parser or a part-of-speech tagger.

Lastly, there is a technical view of the system. The various modules are all connected through TCP/IP sockets and send and receive standardized ASCII messages, with the Broker functioning as an exchange. Because of the Request Broker and the TCP/IP links, the modules can be distributed over multiple machines and multiple platforms in a completely transparent way.

B.3.1 Paradigm Independence

Although initially developed for the KISS Method, the Grammalizer team decided to equip version 2 with a flexible frame structure. This proved to be a good idea when a new version of KISS emerged in the course of 1997, as no code needed to be changed except for the external paradigm specification. The language in which these specifications are written is called *ParaSpec* (Hoppenbrouwers, 1997), of which a brief example is presented here.

```
# ParaSpec example, JH 07-aug-1997.
# This paradigm contains only one template.

paradigm 'Example'
  template 'Action'
    tags <font=helvetica type=roman fg=black bg=white>
      Subject <fg=white bg=black>
      Action <fg=white bg=red>
      Object-Pat <fg=yellow bg=black>
      Prep <fg=green bg=white>
```

```
      Object-Oth <fg=orange bg=black>
    end tags

    frame
      Subject <sem=agent>
      Action primary <sem=action>
      Object-Pat <sem=patient>
      { <max=3>
        Prep <sem=prep>
        Object-Oth <sem=other>
      }
    end frame

    transformation 'Generic Subject'
      Subject := 'somebody'
    end transformation

    transformation 'Generic Object-Pat'
      Object-Pat := 'something'
    end transformation

  end template
end paradigm
```

ParaSpec is a generative grammar that specifies a space of frame types which can be tuned to the analysis paradigm at hand. In the Grammalizer, most modules are sensitive to ParaSpec files and will follow the selected paradigm to a great extent, changing their user interfaces accordingly.

B.3.2 The NLP Modules

The AutoTagger is the main NLP module of the Grammalizer. It consists of several linked programs which together perform the linguistic analysis required. Parts of the system are a rugged text preprocessor to clean up input text and break it up into separate sentences, and a sentence preprocessor to clean up and normalize sentences, preparing them to be fed into the parser. Both these modules work purely on typography; the text preprocessor can be tuned to the particular type of text through a graphical user interface. For example, everything between parentheses is considered a subsentence and is separately presented to the NLP engine.

The linguistic core of the AutoTagger (see Figure B.2) consists of a statistical part of speech tagger (Daelemans et al., 1996) and a version of the AMAZON NL parser (Coppen, 1995), assisted by a Lexicon with pre-established morphosyntax, based on CELEX (Baayen, 1991; Burnage, 1990) which contains roughly 400,000 word forms, 35,000 lemmas, and a compound resolver (see Section 3.3.3). At the moment, all these modules are intended for processing of Dutch sentences only. The part of speech tagger has been trained on a large corpus and produces WOTAN-tagged output, which is subsequently enriched with limited closed-class information

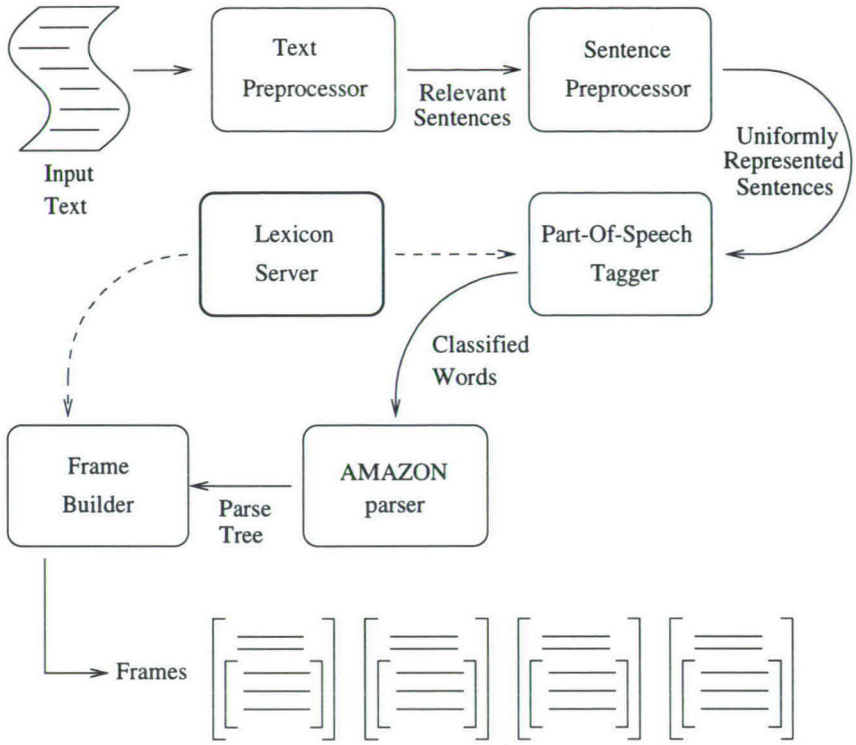

Figure B.2: The AutoTagger Architecture

from the Lexicon. The AMAZON parser tries to parse the result and if successful, produces a parse tree. The Frame Builder inspects the AMAZON parse tree, decides which types of frames can be extracted from it (with the possibility of extracting no frames at all, e.g., when the input sentence did not have parseable contents) and rewrites the resulting frames to normalized form, e.g., verbs in infinitive and nouns in singular. These frames are then sent back to the calling module, typically the Analyzer. The Analyzer subsequently maps the received linguistic frames to KISS primitives through ParaSpec mapping definitions.

Care has been taken to make the previously described AutoTag process as robust as possible, with various checks to insure that no input can break any module and that the resulting frames are complete. We try to adhere to the idea *garbage in, nothing out.*

Appendix C

The TREVI Project

The TREVI (Text Retrieval and Enrichment for Vital Information) Project aims at offering a solution to the problem of *information overflow*, i.e., the difficulty experienced by most organizations in extracting useful information from the large amounts of raw data coming from the numerous electronic textual information services available at local or global level (Internet, proprietary networks, subscription services, World Wide Web, and more). TREVI is a European Committee Esprit Project, EP23311, with various project partners located in Italy, Spain, Germany, England, Belgium, and the Netherlands.

The key result of the TREVI Project will be a set of software tools (the TREVI Toolkit) representing a substantial improvement in the management of distributed textual information sources. The TREVI Toolkit will not rely on simple text-based search tools; instead, it will combine concept-based search and active data mining techniques to enrich online input text streams by providing indexing, abstraction, smart correlation with data and knowledge sources, compilation into electronic publication formats, and subscription capability on the results through communication services (for example, HTML document servers on the WWW).

C.1 TREVI's NLP Architecture

The TREVI parsing module inputs text streams and moves through various stages of linguistic processing, from lexical/morphological through syntactic and semantic analysis supported by a language-dependent Lexicon and Thesaurus Management System (LMS). As a result, the parsing module produces annotated tree structures suitable for the subject identification module. The primary function of the subject identification module is to scan the annotated tree structures in order to determine the subject of the analyzed text streams, based on domain-specific thesaurus concepts retrieved from the LMS.

C.1.1 The LMS

The LMS contains the morphosyntactic knowledge needed for parsing English and Spanish texts, but also allow for easy extension and maintenance of the sublexicons by domain experts (possibly supported by machine-learning tools). The LMS is developed as a separate module with a well-defined interface and query language. For

filling the general part of the LMS, existing lexical tools such as (Euro) WordNet and CELEX were used as much as possible. Additionally, the LMS contains semantic knowledge tuned to specific domains, in order to support TREVI's subject classification process.

In brief, the LMS holds the following information:

Word stems and inflectional forms to support the lemmatization phase of the parsing process.

Extensional object identifiers including geographical names, company names, personal names, dates etc. to support the term identification phase of the parsing process.

Frame descriptions to support the conceptual disambiguation phase following the parsing process.

Thesaurus information including lexical semantics such as embodied in frames, word fields, and qualia structures, to support the subject identification.

The LMS allows for easy extension of its contents by hand as well as by machine, which includes a lexicographer's interface (called the Lexicographer's Workbench, LWB) to background text corpora and the LMS. The storage structure of the LMS is based on feature structures and closely follows the design presented in Chapter 4. The amount of data needed (hundreds of thousands of entries per language) requires a management system comparable to a DBMS. However, a relational DBMS is not very practical—its table and index structure do not fit very well with the feature structure format of lexical items, and DBMSses offer features such as transactions and recovery that are not needed in TREVI. Systems based on feature structures have before been implemented in Prolog (there is even a special Prolog dialect, called LIFE, that combines Prolog and feature structures), but these are academic systems that do not meet industrial standards of robustness and scalability. Therefore, drawing on experience with feature structure systems in LIFE and on top of available OO data management tools, an LMS is being developed that is not only of use in TREVI, but has the potential of begin exploited in other projects as well. It is for the most part implemented in Java.

The LMS actually consists of two modules: a *generator* which compiles LIX LDL definitions into Java classes, and the *database manager* which manages the actual data storage. The compiled LDL modules function as paradigm and language dependent interface layers between the LWB and the independent database manager. The LWB is not considered a technical part of the LMS, due to its client/server architecture, but is obviously closely related.

C.1.2 Lexicon Definitions

TREVI currently supports two languages, English and Spanish. The semantic part of the Lexicon is shared by both languages, but the morphosyntactic part is obviously different. I will first present the complete English definition, including semantics, followed by a partial definition for Spanish morphosyntax. Both definitions are written in the LIX LDL as described in Section 4.4.2. Note that Spanish morphosyntax

has been completely described in TREVI, but space considerations prevent me to include the whole specification in this appendix.

LDL for English

```
# LIX-LDL for English
# By Stijn Hoppenbrouwers
# Last revision 10-jul-1997

# ABSTRACT CLASSES
lexical Word (parent=none forms=root)
tuple Concept (parent=none)
tuple Denotation (parent=none elements=Word,Concept)
#-----------------------------------


# SEMANTICS

# term, roletypes, and frames
tuple Term (parent=Concept)
tuple Role (parent=none)
tuple Marker (elements=Role,EnglishPrep)
 tuple Subject (parent=Role elements=Term)
  tuple Predicate (parent=Role elements=Term)
  tuple Agent (parent=Role elements=Term)
  tuple Patient (parent=Role elements=Term)
  tuple Donator (parent=Role elements=Term)
  tuple Recipient (parent=Role elements=Term)
  tuple Source (parent=Role elements=Term)
  tuple Destination (parent=Role elements=Term)
  tuple Original (parent=Role elements=Term)
  tuple Result (parent=Role elements=Term)

tuple Frame (parent=Concept)
  tuple State (parent=Frame elements=Subject)
  tuple Predication (parent=Frame elements=Subject,Predicate)
  tuple Action (parent=Frame elements=Agent,Patient)
    tuple Transfer (parent=Action elements=Donator,Recipient)
    tuple Transport (parent=Action elements=Source,Destination)
    tuple Transform (parent=Action elements=Original,Result)

#-----------------------------------------------------


# MORPHOSYNTAX

# nouns
```

```
set countN (parent=none type=Concept)
set massN (parent=none type=Concept)
set singTantN (parent=none type=Concept)
set plurTantN (parent=none type=Concept)
set attrN (parent=none type=Concept)
set properN (parent=none type=Concept)
lexical Noun (parent=Word forms=singular,plural)
  lexical EnglishNoun (parent=Noun)
  script EnglishNoun singular
    SCRIPT
    root
    SCRIPT
  script EnglishNoun plural
    SCRIPT
    root + "s"
    SCRIPT

# verbs
set transitiveV (parent=none type=Frame)
set linkV (parent=none type=Frame)
lexical Verb (parent=Word)
  lexical EnglishVerb (parent=Verb forms=inf,imp,1es,2es,3es,1ep,
     2ep,3ep,1as,2as,3as,1ap,2ap,3ap,apart,epart)

  script EnglishVerb inf
    SCRIPT
    root
    SCRIP
  script EnglishVerb imp
    SCRIPT
    root
    SCRIPT
  script EnglishVerb 1es
    SCRIPT
    root
    SCRIPT
  script EnglishVerb 2es
    SCRIPT
    root
    SCRIPT
  script EnglishVerb 3es
    SCRIPT
    root + "s"
    SCRIPT
  script EnglishVerb 1ep
    SCRIPT
    root
```

```
    SCRIPT
  script EnglishVerb 2ep
    SCRIPT
    root
    SCRIPT
  script EnglishVerb 3ep
    SCRIPT
    root
    SCRIPT
  script EnglishVerb 1as
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb 2as
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb 3as
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb 1ap
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb 2ap
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb 3ap
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb apart
    SCRIPT
    root + "ed"
    SCRIPT
  script EnglishVerb epart
    SCRIPT
    root + "ing"
    SCRIPT

    lexical EnglishComplexVerb (parent=EnglishVerb forms=Particle)
  script EnglishComplexVerb inf
    SCRIPT
    root + " " + Particle
    SCRIPT
```

```
script EnglishComplexVerb imp
  SCRIPT
  root + " " + Particle
  SCRIPT
script EnglishComplexVerb 1es
  SCRIPT
  root + " " + Particle
  SCRIPT
script EnglishComplexVerb 2es
  SCRIPT
  root + " " + Particle
  SCRIPT
script EnglishComplexVerb 3es
  SCRIPT
  root + "s " Particle
  SCRIPT
script EnglishComplexVerb 1ep
  SCRIPT
  root + " " + Particle
  SCRIPT
script EnglishComplexVerb 2ep
  SCRIPT
  root + " " + Particle
  SCRIPT
script EnglishComplexVerb 3ep
  SCRIPT
  root + " " + Particle
  SCRIPT
script EnglishComplexVerb 1as
  SCRIPT
  root + "ed " + Particle
  SCRIPT
script EnglishComplexVerb 2as
  SCRIPT
  root + "ed " + Particle
  SCRIPT
script EnglishComplexVerb 3as
  SCRIPT
  root + "ed " + Particle
  SCRIPT
script EnglishComplexVerb 1ap
  SCRIPT
  root + "ed " + Particle
  SCRIPT
script EnglishComplexVerb 2ap
  SCRIPT
  root + "ed " + Particle
```

```
    SCRIPT
  script EnglishComplexVerb 3ap
    SCRIPT
    root + "ed " + Particle
    SCRIPT
  script EnglishComplexVerb apart
    SCRIPT
    root + "ed " + Particle
    SCRIPT
  script EnglishComplexVerb epart
    SCRIPT
    root + "ing " + Particle
    SCRIPT
#-----------------------------------

# modifiers
set attrMod (parent=none type=Frame)
set predMod (parent=none type=Frame)
set postposMod (parent=none type=Frame)
set noAdvMod (parent=none type=Modifier)
set noAdjMod (parent=none type=Modifier)
set bothMod (parent=none type=Modifier)
conjunction adj (parent=none sets=noAdvMod,bothMod)
conjunction adv (parent=none sets=noAdjMod,bothMod)
lexical Modifier (parent=Word forms=root)
  lexical EnglishModifier (parent=Modifier forms=adj,com,sup,adv)
  script EnglishModifier adj
    SCRIPT
    root
    SCRIPT
  script EnglishModifier com
    SCRIPT
    root + "er"
    SCRIPT
  script EnglishModifier sup
    SCRIPT
    root + "est"
    SCRIPT
  script EnglishModifier adv
    SCRIPT
    root + "ly"
    SCRIPT

# numerals
lexical Numeral (parent=Word forms=root)
  lexical EnglishNumeral (parent=Numeral forms=card,ord,cardNum,ordNum)
```

```
# articles
lexical Article (parent=Word forms=root)
  lexical EnglishArticle (parent=Article)

# pronouns
lexical Pronoun (parent=Word forms=root)
  set mascPron (parent=none type=Pronoun)
  set femPron (parent=none type=Pronoun)
  set 1pPron (parent=none type=Pronoun)
  set 2pPron (parent=none type=Pronoun)
  set 3pPron (parent=none type=Pronoun)
  set singPron (parent=none type=Pronoun)
  set plurPron (parent=none type=Pronoun)
  set whPron (parent=none type=Pronoun)
  set detPron (parent=none type=Pronoun)
  set pronPron (parent=none type=Pronoun)
  lexical EnglishPronoun (parent=Pronoun forms=pers,dem,poss,refl)

# Conjunction
lexical Conjunction (parent=Word forms=root)
  set coordConj (parent=none type=Conjunction)
  set subordConj (parent=none type=Conjunction)
  lexical EnglishConjunction (parent=Conjunction)

# Interjection
lexical Interjection (parent=Word forms=root)
  lexical EnglishInterjection (parent=Interjection)

# Preposition
lexical Preposition (parent=Word forms=root)
  lexical EnglishPreposition (parent=Preposition)

# Letters
lexical Letter (parent=Word forms=root)

# Abbreviations
lexical Abbreviation (parent=Word forms=root)
  lexical EnglishAbbreviation (parent=Abbreviation)

# To
lexical EnglishTo (parent=Word forms=root)
```

LDL for Spanish

```
# Partial specification for Spanish
# By Ellen-Petra Kester
# Last revision 20-aug-1997

# \ followed by accent has the meaning of character with accent on top
  .
  .
  .
# MORPHOSYNTAX

# nouns
lexical Noun (parent=Word)
lexical SpanishNoun (parent=Noun forms=singular,plural)
  script SpanishNoun singular
    SCRIPT
    root
    SCRIPT
  script SpanishNoun plural
    SCRIPT
    if vowel(tail(root,1)) then plural=root + "s"
    if consonant(tail(root,1)) then plural=root + "es"
    SCRIPT

#modifiers
  lexical Modifier (parent=Word)
  lexical SpanishModifier (parent=Modifier
                    forms=massinadj,femsinadj,maspluadj,fempluadj,adv)
    script SpanishModifier massinadj
      SCRIPT
      root
      SCRIPT
    script SpanishModifier femsinadj
      SCRIPT
      if vowel(tail(root,1),"o") then femsinadj = tailer(root,1)  + "a"
      else femsinadj = root
      SCRIPT
    script SpanishModifier maspluadj
      SCRIPT
      if vowel(tail(root,1))  then maspluadj=root + "s"
      if consonant(tail(root,1)) then maspluadj=root + "es"
      SCRIPT
    script SpanishModifier fempluadj
      SCRIPT
      if vowel(tail(root,1),"o") then fempluadj= tailer(root,1) + "as"
      else if consonant(tail(root,1)) then fempluadj=root + "es"
```

```
      else fempluadj=root + "s"
      SCRIPT

    set noAdvSpanMod (parent=none type=Modifier)
    set noAdjSpanMod (parent=none type=Modifier)
    set bothSpanMod (parent=none type=Modifier)
    conjunction adj (parent=none sets=noAdvSpanMod,bothSpanMod)
    conjunction adv (parent=none sets=noAdjSpanMod,bothSpanMod)

# numerals
lexical numeral (parent=Word)
  lexical SpanishNumeral (parent=Numeral forms=card,ord,frac)

# articles
lexical Article (parent=Word)
lexical SpanishArticle (parent=Article forms=masart,femart)

# pronouns
lexical Pronoun (parent=Word
forms=massinpron,femsinpron,masplupron,femplupron,neutpron)
  set maleSpanPron (parent=none type=Pronoun)
  set femaleSpanPron (parent=none type=Pronoun)
  set 1pSpanPron (parent=none type=Pronoun)
  set 2pSpanPron (parent=none type=Pronoun)
  set 3pSpanPron (parent=none type=Pronoun)
  set singSpanPron (parent=none type=Pronoun)
  set pluSpanPron (parent=none type=Pronoun)
  set whSpanPron (parent=none type=Pronoun)
  set detSpanPron (parent=none type=Pronoun)
  set pronSpanPron (parent=none type=Pronoun)
  lexical SpanishPronoun (parent=Pronoun forms=pers,dem,poss,refl)

# conjunction
lexical Conjunction (parent=Word)
  lexical SpanishConjunction (parent=Conjunction)

# interjection
lexical Interjection (parent=Word)
  lexical SpanishInterjection (parent=Interjection)

# preposition
lexical Preposition (parent=Word)
  lexical SpanishPreposition (parent=Preposition)


#Specification of verb1 features
```

```
# verbs
lexical Verb (parent=Word)
  lexical SpanishVerb (parent=Verb forms=
  1sinpresind,2sinpresind,3sinpresind,1plupresind,2plupresind,3plupresind,
  1sinpressub,2sinpressub,3sinpressub,1plupressub,2plupressub,3plupressub,
  1sinimpfind,2sinimpfind,3sinimpfind,1pluimpfind,2pluimpfind,3pluimpfind,
  1sinimpfsub,2sinimpfsub,3sinimpfsub,1pluimpfsub,2pluimpfsub,
  3pluimpfsub,
  1sinpret,2sinpret,3sinpret,1plupret,2plupret,3plupret,
  1sinfut,2sinfut,3sinfut,1plufut,2plufut,3plufut,
  1sincond,2sincond,3sincond,1plucond,2plucond,3plucond,
  2sinimpa,3sinimpa,2pluimpa,3pluimpa,
  inf,
  gerund,
  part,
  gerundclit3sinfemacc, gerundclit3plufemacc, gerundclit3sinfemdat,
  gerundclit3sinmasdat, gerundclit3sinmasacc, gerundclit3plufemdat,
  gerundclit3plumasdat, gerundclit3plumasacc,
  gerundclit3sinneutacc,
  gerundclit1sin, gerundclit1sin_3sinfemacc, gerundclit1sin_3plufemacc,
  gerundclit1sin_3sinfemdat, gerundclit1sin_3sinmasdat,
  gerundclit1sin_3sinmasacc, gerundclit1sin_3plufemdat,
  gerundclit1sin_3plumasdat, gerundclit1sin_3plumasacc,
  gerundclit1sin_3sinneutacc,
  gerundclit1plu, gerundclit1plu_3sinfemacc, gerundclit1plu_3plufemacc,
  gerundclit1plu_3sinfemdat, gerundclit1plu_3sinmasdat,
  gerundclit1plu_3sinmasacc, gerundclit1plu_3plufemdat,
  gerundclit1plu_3plumasdat, gerundclit1plu_3plumasacc,
  gerundclit1plu_3sinneutacc,
  gerundclit2plu, gerundclit2plu_3sinfemacc,
  gerundclit2plu_3plufemacc, gerundclit2plu_3sinfemdat,
  gerundclit2plu_3sinmasdat, gerundclit2plu_3sinmasacc,
  gerundclit2plu_3plufemdat, gerundclit2plu_3plumasdat,
  gerundclit2plu_3plumasacc,
  gerundclit2plu_3sinneutacc,
  gerundclit3sin, gerundclit3sin_3sinfemacc,  gerundclit3sin_3plufemacc,
  gerundclit3sin_3sinfemdat, gerundclit3sin_3sinmasdat,
  gerundclit3sin_3sinmasacc, gerundclit3sin_3plufemdat,
  gerundclit3sin_3plumasdat, gerundclit3sin_3plumasacc,
  gerundclit3sin_3sinneutacc,gerundclit3sin_1sin,
  gerundclit3sin_1sin_3sinfemacc, gerundclit3sin_1sin_3plufemacc,
  gerundclit3sin_1sin_3sinfemdat,
  gerundclit3sin_1sin_3sinmasdat, gerundclit3sin_1sin_3sinmasacc,
  gerundclit3sin_1sin_3plufemdat, gerundclit3sin_1sin_3plumasdat,
  gerundclit3sin_1sin_3plumasacc, gerundclit3sin_1sin_3sinneutacc,
  gerundclit3sin_1plu, gerundclit3sin_1plu_3sinfemacc,
  gerundclit3sin_1plu_3plufemacc, gerundclit3sin_1plu_3sinfemdat,
```

gerundclit3sin_1plu_3sinmasdat, gerundclit3sin_1plu_3sinmasacc,
gerundclit3sin_1plu_3plufemdat, gerundclit3sin_1plu_3plumasdat,
gerundclit3sin_1plu_3plumasacc, gerundclit3sin_1plu_3sinneutacc,
gerundclit3sin_2plu, gerundclit3sin_2plu_3sinfemacc,
gerundclit3sin_2plu_3plufemacc, gerundclit3sin_2plu_3sinfemdat,
gerundclit3sin_2plu_3sinmasdat, gerundclit3sin_2plu_3sinmasacc,
gerundclit3sin_2plu_3sinneutacc, gerundclit3sin_2plu_3plumasacc,
gerundclit3sin_2sin, gerundclit3sin_2sin_3sinfemacc,
gerundclit3sin_2sin_3plufemacc, gerundclit3sin_2sin_3sinfemdat,
gerundclit3sin_2sin_3sinmasdat, gerundclit3sin_2sin_3sinmasacc,
gerundclit3sin_2sin_3plufemdat, gerundclit3sin_2sin_3plumasdat,
gerundclit3sin_2sin_3sinneutacc, gerundclit3sin_2sin_3plumasacc,
gerundclit2sin, gerundclit2sin_3sinfemacc, gerundclit2sin_3plufemacc,
gerundclit2sin_3sinfemdat, gerundclit2sin_3sinmasdat,
gerundclit2sin_3sinmasacc, gerundclit2sin_3plufemdat,
gerundclit2sin_3plumasdat, gerundclit2sin_3plumasacc,
gerundclit2sin_3sinneutacc,
infclit3sinfemacc, infclit3plufemacc, infclit3sinfemdat,
infclit3sinmasdat, infclit3sinmasacc, infclit3plufemdat,
infclit3plumasdat, infclit3plumasacc, infclit3sinneutacc,
infclit1sin, infclit1sin_3sinfemacc, infclit1sin_3plufemacc,
infclit1sin_3sinfemdat, infclit1sin_3sinmasdat,
infclit1sin_3sinmasacc, infclit1sin_3plufemdat,
infclit1sin_3plumasdat,
infclit1sin_3sinneutacc, infclit1sin_3plumasacc,
infclit1plu, infclit1plu_3sinfemacc, infclit1plu_3plufemacc,
infclit1plu_3sinfemdat, infclit1plu_3sinmasdat,
infclit1plu_3sinmasacc, infclit1plu_3plufemdat,
infclit1plu_3plumasdat, infclit1plu_3plumasacc,
infclit1plu_3sinneutacc,
infclit2plu, infclit2plu_3sinfemacc,
infclit2plu_3plufemacc, infclit2plu_3sinfemdat,
infclit2plu_3sinmasdat, infclit2plu_3sinmasacc,
infclit2plu_3plufemdat, infclit2plu_3plumasdat,
infclit2plu_3sinneutacc, infclit2plu_3plumasacc,
infclit3sin, infclit3sin_3sinfemacc, infclit3sin_3plufemacc,
infclit3sin_3sinfemdat, infclit3sin_3sinmasdat,
infclit3sin_3sinmasacc, infclit3sin_3plufemdat,
infclit3sin_3plumasdat, infclit3sin_3sinneutacc,
infclit3sin_3plumasacc, infclit3sin_1sin,
infclit3sin_1sin_3sinfemacc, infclit3sin_1sin_3plufemacc,
infclit3sin_1sin_3sinfemdat,
infclit3sin_1sin_3sinmasdat, infclit3sin_1sin_3sinmasacc,
infclit3sin_1sin_3plufemdat, infclit3sin_1sin_3plumasdat,
infclit3sin_1sin_3sinneutacc, infclit3sin_1sin_3plumasacc,
infclit3sin_1plu_3sinmasdat, infclit3sin_1plu_3sinmasacc,
infclit3sin_1plu_3plufemdat, infclit3sin_1plu_3plumasdat,

```
  infclit3sin_1plu_3sinneutacc, infclit3sin_1plu_3plumasacc,
  infclit3sin_1plu, infclit3sin_2plu_3sinfemacc,
   infclit3sin_2plu_3plufemacc, infclit3sin_2plu_3sinfemdat,
  infclit3sin_2plu_3sinmasdat,
  infclit3sin_2plu_3plufemdat, infclit3sin_2plu_3plumasdat,
  infclit3sin_2plu_3plumasacc, infclit3sin_2plu_3sinmasacc,
  infclit3sin_2plu_3sinneutacc,
  infclit3sin_2sin, infclit3sin_2sin_3sinfemacc,
  infclit3sin_2sin_3plufemacc, infclit3sin_2sin_3sinfemdat,
  infclit3sin_2sin_3sinmasdat, infclit3sin_2sin_3sinmasacc,
  infclit3sin_2sin_3plufemdat, infclit3sin_2sin_3plumasdat,
  infclit3sin_2sin_3sinneutacc, infclit3sin_2sin_3plumasacc,
  infclit2sin,  infclit2sin_3sinfemacc, infclit2sin_3plufemacc,
  infclit2sin_3sinfemdat, infclit2sin_3sinmasdat,
  infclit2sin_3sinmasacc, infclit2sin_3plufemdat,
  infclit2sin_3plumasdat,
  infclit2sin_3sinneutacc,
  infclit2sin_3plumasacc)

  lexical SpanishVerb1(parent=SpanishVerb)

#primera conjugacion, presente, indicativo

  script SpanishVerb1 1sinpresind
    SCRIPT
    root + "o"
    SCRIPT

  script SpanishVerb1 2sinpresind
    SCRIPT
    root + "as"
    SCRIPT

  script SpanishVerb1 3sinpresind
    SCRIPT
    root + "a"
    SCRIPT

  script SpanishVerb1 1plupresind
    SCRIPT
    root + "amos"
    SCRIPT

  script SpanishVerb1 2plupresind
    SCRIPT
    root + "\'ais"
    SCRIPT
```

```
  script SpanishVerb1 3plupresind
    SCRIPT
    root + "an"
    SCRIPT

#primera conjugacion, presente, subjuntivo

  script SpanishVerb1 1sinpressub
    SCRIPT
    root + "e"
    SCRIPT

  script SpanishVerb1 2sinpressub
    SCRIPT
    root + "es"
    SCRIPT

  script SpanishVerb1 3sinpressub
    SCRIPT
    root + "e"
    SCRIPT

  script SpanishVerb1 1plupressub
    SCRIPT
    root + "emos"
    SCRIPT

  script SpanishVerb1 2plupressub
    SCRIPT
    root + "\'eis"
    SCRIPT

  script SpanishVerb1 3plupressub
    SCRIPT
    root + "en"
    SCRIPT

#primera conjugacion, imperfecto, indicativo

  script SpanishVerb1 1sinimpfind
    SCRIPT
    root + "aba"
    SCRIPT

  script SpanishVerb1 2sinimpfind
    SCRIPT
```

```
    root + "abas"
    SCRIPT

  script SpanishVerb1 3sinimpfind
    SCRIPT
    root + "aba"
    SCRIPT

  script SpanishVerb1 1pluimpfind
    SCRIPT
    root + "\'abamos"
    SCRIPT

  script SpanishVerb1 2pluimpfind
    SCRIPT
    root + "abais"
    SCRIPT

  script SpanishVerb1 3pluimpfind
    SCRIPT
    root + "aban"
    SCRIPT

#primera conjugacion, imperfecto, subjuntivo

  script SpanishVerb1 1sinimpfsub
    SCRIPT
    root + "ara"
    root + "ase"
    SCRIPT

  script SpanishVerb1 2sinimpfsub
    SCRIPT
    root + "aras"
    root + "ases"
    SCRIPT

  script SpanishVerb1 3sinimpfsub
    SCRIPT
    root + "ara"
    root + "ase"
    SCRIPT

  script SpanishVerb1 1pluimpfsub
    SCRIPT
    root + "\'aramos"
    root + "\'asemos"
```

```
    SCRIPT

  script SpanishVerb1 2pluimpfsub
    SCRIPT
    root + "arais"
    root + "aseis"
    SCRIPT

  script SpanishVerb1 3pluimpfsub
    SCRIPT
    root + "aran"
    root + "asen"
    SCRIPT
    .
    .
    .

#Clitics attached to the suffixes of gerunds and infinitives

# GERUNDS

#primera conjugacion, gerundio con clitico 3sinfemacc
  script SpanishVerb1 gerundclit3sinfemacc
    SCRIPT
    root + "\'andola"
    SCRIPT

#primera conjugacion, gerundio con clitico 3plufemacc
  script SpanishVerb1 gerundclit3plufemacc
    SCRIPT
    root + "\'andolas"
    SCRIPT

#primera conjugacion, gerundio con clitico 3sinfemdat
  script SpanishVerb1 gerundclit3sinfemdat
    SCRIPT
    root + "\'andole"
    SCRIPT

#primera conjugacion, gerundio con clitico 3sinmasdat
  script SpanishVerb1 gerundclit3sinmasdat
    SCRIPT
    root + "\'andole"
    SCRIPT

#primera conjugacion, gerundio con clitico 3sinmasacc
  script SpanishVerb1 gerundclit3sinmasacc
```

```
    SCRIPT
    root + "\'andole"
    root + "\'andolo"
    SCRIPT

#primera conjugacion, gerundio con clitico 3plufemdat
  script SpanishVerb1 gerundclit3plufemdat
    SCRIPT
    root + "\'andoles"
    SCRIPT

#primera conjugacion, gerundio con clitico 3plumasdat
  script SpanishVerb1 gerundclit3plumasdat
    SCRIPT
    root + "\'andoles"
    SCRIPT

#primera conjugacion, gerundio con clitico 3plumasacc
  script SpanishVerb1 gerundclit3plumasacc
    SCRIPT
    root + "\'andoles"
    SCRIPT

#primera conjugacion, gerundio con clitico 3sinmasacc

#primera conjugacion, gerundio con clitico 3sinneutacc
  script SpanishVerb1 gerundclit3sinneutacc
    SCRIPT
    root + "\'andolo"
    SCRIPT

#primera conjugacion, gerundio con clitico 3plumasacc
  script SpanishVerb1 gerundclit3plumasacc
    SCRIPT
    root + "\'andolos"
    SCRIPT
  .
  .
  .
```

C.1.3 Lexicon Contents

To fill the LMS, we used Aries, CELEX and WordNet resources. Using various extraction and transformation programs, mainly written in Tcl and UNIX shell scripts, the raw input material was transformed into several clean ASCII[1] files that can be read by the LMS loader.

We performed a matching operation between CELEX and WordNet to combine CELEX's morphosyntactic information with WordNet's semantic and thesaurus information. We will attempt a comparable match between Aries and WordNet, if possible using EuroWordNet Spanish thesaurus information once it becomes available. Specific domain information for both languages will be added using text corpora from the respective domains, and if all else fails, manually. Frame specifications for the most significant verbs and cognitive schemata for nominal word fields will be added by hand as well.

[1]Care has been taken to insure correct treatment of diacritics.

Samenvatting

Eén van de grootste uitdagingen bij het ontwikkelen van informatie- en communicatiesystemen is het op elkaar afstemmen van de wensen van de gebruikers en de eigenschappen van het te bouwen systeem. De wetenschap van de *conceptuele modellering* probeert deze afstemming te verbeteren door systeembeschrijvingen op te stellen op een abstract niveau, dichter bij de leefwereld van de gebruiker (zijn of haar *domein*) en verder weg van de machine.

Dit proefschrift behandelt één van de mogelijke manieren waarop systeembeschrijvingen dichter bij de gebruiker kunnen worden gebracht, namelijk het gebruiken van natuurlijke taal. De menselijke talen, zoals Engels of Nederlands, zijn bij uitstek geschikt voor het communiceren over domeinen, en zouden dus ook moeten kunnen bijdragen aan geschikte domeinbeschrijvingen. Communicatieuitingen in taal zijn gefundeerd in woorden, en deze woorden dragen betekenis. Filosofen zoals Hamann, Wittgenstein, Heidegger, Foucault, Frege en Peirce stellen dat woorden (en dus ook taal) de *essentie* zijn van zowel kennis als menselijk gedrag. Dit proefschrift probeert de vraag te beantwoorden hoe de in domeinterminologie aanwezige impliciete kennis kan worden ingezet om de conceptuele analyse van een domein te verbeteren. Daarna behandelt het een aantal methoden en technieken om rondom een *Lexicon*, een gecomputeriseerde kennisbank gevuld met woorden, ondersteunende systemen te bouwen die analisten en programmeurs helpen bij hun uiteindelijke taak, het ontwerpen en construeren van een informatie- en communicatiesysteem.

In tegenstelling tot een *Data Dictionary*, die vooral op menselijk gebruik gericht is en dus veelal omschrijvingen in natuurlijke taal bevat, aangevuld met technische eigenschappen van het betreffende woord, is een Lexicon met name gericht op machinaal gebruik. Ook bevat een Lexicon typisch geen omschrijvingen of definities van woorden, en geen domeinspecifieke details. In deze zin vullen een Lexicon en een Data Dictionary elkaar aan; het Lexicon levert de taalkundige gegevens van woorden (semantische relaties en morfosyntaxis) en de Data Dictionary de domeinspecifieke definities.

Vooral bij het opstellen van een programma van eisen (*requirements engineering*) is natuurlijke taal een zeer waardevol hulpmiddel gebleken. In de vroege fases van de meeste projecten wordt veel informatie uitgewisseld in de vorm van taal, en taal blijft voor veel betrokkenen de enige manier om met elkaar te communiceren (technische tekeningen en wiskundige beschrijvingen zijn vaak niet geschikt). Een Lexicon kan al in deze fases van het project ondersteuning bieden als centraal opslagpunt van (domein)terminologie. Een aantal communicatiestoornissen kunnen hiermee al worden opgelost, zoals spraakverwarring over de betekenis van synoniemen en homoniemen. Wanneer het Lexicon tevens een groot aantal algemene woorden bevat,

kunnen controles worden uitgevoerd op de aannemelijkheid van sommige modelconstructies. Ook kan de modelterminologie worden gebruikt om verschillende modellen met elkaar te vergelijken en overlappingen en discrepanties te detecteren. Dit kan nuttig zijn voor hergebruik van bestaande modellen en wanneer twee of meerdere modellen verenigd moeten worden, bijvoorbeeld bij bedrijfsfusies.

In de computationele taalkunde is het gebruik van gecomputeriseerde Lexicons wijd verbreid. Een aantal systemen bevat vele tienduizenden woorden, verkregen door handmatige verzameling of door machinale verwerking van bestaande woordenboeken. Vaak is er ook een structuur in de woorden aangebracht die uitstijgt boven een alfabetisch-lexicografische ordening. De wetenschap is er nog niet in geslaagd om een eenduidige structuur te ontwerpen die aan alle eisen tegemoet komt, reden waarom in dit proefschrift de nadruk wordt gelegd op een *flexibel* Lexicon waarvan de structuur naar behoefte aangepast kan worden.

Eén van de pijlers van het Lexicon zoals voorgesteld in dit proefschrift is de scheiding tussen *concept* en *lexeem*, ook wel *lexical* genoemd. Dit is volledig in lijn met ideeën van bijvoorbeeld De Saussure, Aristoteles, Ogden en Richards en Peirce, maar de wijze waarop deze scheiding is geïmplementeerd in een objectgeoriënteerde structuur is vernieuwend. Door gebruik te maken van overerving tussen lexicals en een combinatie van gestipuleerde vormen en algorithmische afleidingsregels kan het lexicale deel van het Lexicon op een gestructureerde manier worden opgezet, met een flinke reductie van redundante informatie. Een dergelijke opzet kan ook worden gemaakt voor het conceptuele, semantische deel van het Lexicon, waarbij de nadruk zou moeten komen te liggen op de *leerbaarheid* van nog onbekende concepten door een gebruiker. Verscheidene categoriesystemen worden gepresenteerd, ieder met de eigen voor- en nadelen. Een werkbaar Lexicon zou in ieder geval *analytische* (top-down) en *pragmatische* (bottom-up) hierarchieën moeten ondersteunen om alle gebruikersgroepen te kunnen bedienen. Daarnaast moeten subsets van de opgeslagen terminologie kunnen worden onderkend om verschillende domeinen uit elkaar te kunnen houden.

Vervolgens wordt een implementeerbaar ontwerp van een Lexicon Management Systeem beschreven. Een dergelijk systeem is opgebouwd uit drie lagen, die ieder een specifiek deel van het werk voor hun rekening nemen:

1. De opslaglaag, waarin de feitelijke opslag van alle Lexicongegevens plaatsvindt, maar die voor de gebruikers nauwelijks ter zake doet.
2. De paradigma/taallaag, die bepaalt volgens welk taalkundig paradigma de gebruikers de opgeslagen gegevens kunnen manipuleren en van welke verschillende talen er morfosyntactische informatie kan worden opgevraagd.
3. De lexicografische laag, die door de gebruikers gemanipuleerd kan worden en waarin de opgeslagen semantische en morfosyntactische informatie georganiseerd wordt.

Al deze drie lagen worden technisch opgebouwd uit objecten die gegroepeerd zijn in een beperkt aantal klassen, die ieder voor zich gespecialiseerd zijn in hun specifieke taak. Hierbij is aandacht besteed aan efficiënte implementeerbaarheid op een computersysteem, maar zoveel mogelijk zonder ook werkelijk implementatievoorstellen te doen. Als voorbeeld wordt een mogelijke Lexicondefinitie gegeven met als

paradigma Functionele Grammatica en als taal het Engels. Het hoofdstuk besluit met een uitgebreide beschrijving van LIX, een technische communicatietaal tussen het Lexicon Management Systeem en de gebruikers (waarbij gebruikers voornamelijk machines zullen zijn).

Daarna komen twee praktische toepassingen van een Lexicon en lexicale modelanalyse aan bod. De NIAM-methode voor conceptuele datamodellering wordt beschreven en daarna vergeleken met een taalgeoriënteerde benadering. Daaruit blijkt dat NIAM relatief weinig aandacht schenkt aan de feitelijke betekenis van taaluitingen en zich meer richt op een technisch correcte weergave van relevante stukjes kennis over gegevensstructuren in het domein. De auteur doet een aantal aanbevelingen die NIAM-analyses beter zouden moeten laten aansluiten bij de cognitieve intuïties van niet-technici, door nadrukkelijk de taaluitingen voorop te stellen en dan initieel een technisch minder complete analyse te produceren. Deze analyse is dan later relatief eenvoudig om te zetten naar een technisch complete en correcte analyse terwijl de originele taaluitingen grotendeels behouden kunnen blijven.

De KISS-methode voor conceptuele actiemodellering richt zich van nature al op taaluitingen, en sluit dan ook goed aan bij de voorgestelde lexicale benadering. Door een rigoreuze lexicale analyse komen echter kleine onvolkomendheden in de methode aan de oppervlakte die op een eenvoudige manier kunnen worden weggewerkt als naar de theoretische basis van woorden – vooral werkwoorden – wordt gekeken. KISS omvat een aantal kennisrepresentatieaspecten die direct verband houden met de organisatie van concepten in het Lexicon, en kan dus ook profiteren van reeds beschikbare kennis in het Lexicon. Verscheidene taalanalysetechnieken kunnen ook inzicht verschaffen in normaliter minder intuïtieve aspecten van KISS, zoals meronymie en objectrollen. Een aantal technieken zijn geïmplementeerd in het *Grammalizer*-project, waarbij noties uit de taalanalyse worden vertaald naar specifieke KISS-concepten.

Het boek besluit met een overzicht van implementaties van lexica, gebaseerd op werk van derden en op het werk van de auteur zelf.

Bibliography

Aarts, F. and Aarts, J. (1986). *English Syntactic Structures.* Pergamon Institute of English and Martinus Nijhoff.

Aarts, J. M. G. (1976). *Adjective-Noun Combinations: A Model for their Semantic Interpretation.* PhD thesis, University of Nijmegen.

Abbott, R. J. (1987). Knowledge abstraction. *Communications of the ACM*, 30(8):664–671.

Abrial, J. (1974). Data semantics. In Klimbe, J. and Koffeman, K., editors, *Database Management Systems.* Elsevier North Holland, New York.

Aït-Kaci, H. (1986). Type subsumption as a model of computation. In Kerschberg, L., editor, *Expert Database Systems*, pages 115–139. Benjamin/Cummings, Menlo Park, California.

Amsler, R. and White, J. (1979). Development of a computational methodology for deriving natural language semantic structures via analysis of machine-readable dictionaries. Technical Report MCS77-01315, National Science Foundation.

Anderson, R. and Freebody, P. (1979). *Vocabulary Knowledge and Reading.* Urbana-Champaign, IL: Center for the Study of Reading.

Anthes, G. (1994). Creep: De kwelling van de achteraf toegevoegde eisen. *Computable*, 27(29/30):15–17.

Asymetrix (1994). *InfoModeler, A Guide to FORML.* Asymetrix Corporation, 110, 110th Avenue NE, Suite 700, Bellevue, WA 98004, Washington.

Austin, J. (1962). *How to Do Things with Words.* Clarendon Press, Oxford.

Baayen, R. (1991). De CELEX Lexicale Databank. *Forum der Letteren*, 32:221–231.

Bach, K. and Harnish, R. (1979). *Linguistic Communication and Speech Acts.* MIT Press, Cambridge MA.

Backus, J. (1959). The Syntax and Semantics of the Proposed International Algebraic Language of the Zürich ACM-GAMM Conference. In *Proceedings International Conference on Information Processing, UNESCO, Paris*, pages 125–132.

Beedham, C. and Bloor, M. (1989). English for computer science and the formal realization of communicative functions. *Fachsprache*, 11(2):13–23.

Berlin, B. (1972). Speculations on the growth of ethnobotanical nomenclature. In *Language in Society*, pages 51–86. Cambridge University Press, London.

Berlin, B. and Kay, P. (1969). *Basic Color Terms: Their Universality and Evolution.* University of California Press, Berkeley and Los Angeles.

Bloomfield, L. (1933). *Language.* Allen & Unwin, London.

Boguraev, B. and Briscoe, T., editors (1989). *Computational Lexicography for Natural Language Processing.* Longman.

Booch, G. (1994). *Object-Oriented Analysis and Design (with Applications).* The Benjamin/Cummings Publishing Company, Inc., Santa Clara, California.

Bouzeghoub, M. and Métais, E., editors (1995). *First International Workshop on Applications of Natural Language to Data Bases.* Laboratoire PRISM, AFCET.

Briscoe, T. (1991). Lexical Issues in Natural Language Processing. In Klein, E. and Veltman, F., editors, *Natural Language and Speech*, pages 39–68. Springer-Verlag.

Bunt, H. (1981). *The Formal Semantics of Mass Terms.* PhD thesis, University of Amsterdam.

Burg, J. (1996). *Linguistic Instruments in Requirements Engineering.* PhD thesis, Vrije Universiteit, Amsterdam. ISBN 90 5199 316 1 (IOS Press), ISBN 4 274 90144 0 C3000 (Ohmsha).

Burnage, G. (1990). *CELEX: a guide for users.* CELEX Centre for Lexical Information, Max Planck Institute for Psycholinguistics, Nijmegen, The Netherlands.

Calzolari, N. (1988). The dictionary and the thesaurus can be combined. In Evens, M. W., editor, *Relational Models of the Lexicon*, pages 73–96. Cambridge University Press.

Calzolari, N., editor (1993a). *Course in Computational Lexicons.* Fifth European Summer School in Logic, Language, and Meaning, Faculdade de Letras, Universidade de Lisboa.

Calzolari, N. (1993b). Detecting patterns in a lexical database. In *Reader, Fifth European Summer School in Logic, Language, and Information.*

Carpenter, R. (1990). Typed Feature Structures: Inheritance, (In)equality, and Extensionality. In *Proceedings of the Workshop on Inheritance in Natural Language Processing, Tilburg*, pages 9–18.

Chaffin, R., Herrmann, D., and Winston, M. (1988). An empirical taxonomy of part-whole relations: Effects of part-whole type on relation identification. *Language and Cognitive Processes*, 1(1):9–36.

Chen, P.-S. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36.

Chen, P.-S. (1983). English Sentence Structure and Entity-Relationship Diagrams. *Information Systems*, 29:127–149.

Copestake, A., Sanfilippo, A., Briscoe, T., and de Paiva, V. (1992). The ACQUILEX LKB: an Introduction. In Briscoe, T., Copestake, A., and de Paiva, V., editors, *Default Inheritance in Unification Based Approaches to the Lexicon*. Unknown publisher.

Coppen, P. (1995). A new version of the AMAZON/CASUS system. In *Proceedings of the Department of Language and Speech*, volume 18, pages 85–90. Nijmegen University.

Cowie, J. and Lehnert, W. (1996). Information Extraction. *Communications of the ACM*, 39(1):80–91.

Daelemans, W. (1990). Inheritance in Object-Oriented Natural Language Processing. In Daelemans, W. and Gazdar, G., editors, *Inheritance in Natural Language Processing*, pages 30–38. Institute for Language Technology and Artificial Intelligence, Tilburg University, The Netherlands.

Daelemans, W. and Gazdar, G., editors (1990). *Inheritance in Natural Language Processing*, Tilburg University, The Netherlands. Institute for Language Technology and Artificial Intelligence.

Daelemans, W., Smedt, K. D., and Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics*, 18(2):205–218.

Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996). MBT: A Memory-Based Part of Speech Tagger-Generator. In *Proceedings of the WVLC, Copenhagen.*

de Saussure, F. (1916). *Cours de linguistique générale.* Payot, Paris.

de Troyer, O. (1991). The oo-binary relationship model: A truly object-oriented conceptual model. In Andersen, R., jr., J. B., and Sølvberg, A., editors, *Lecture Notes in Computer Science 498, "Advanced Information Systems Engineering"*. Springer-Verlag.

de Troyer, O., Meersman, R., and Ponsaert, F. (1983). *RIDL User Guide.*

de Troyer, O. M. F. (1993). *On Data Schema Transformation.* PhD thesis, Tilburg University, the Netherlands.

Dedene, G. and Snoeck, M. (1994). MERODE: A Model-driven Entity-Relationship Object-oriented DEvelopment model. *ACM SIGSOFT Software Engineering Notes*, 13(3):51–61.

Derksen, C., Frederiks, P., and van der Weide, T. (1996). Paraphrasing as a Technique to Support Object-Oriented Analysis. In van de Riet, R. et al., editors, *Applications of Natural Language to Information Systems*, pages 28–39.

Dietz, J. (1990). A communication oriented approach to conceptual modelling of information systems. In Steinholz, B., Sølvberg, A., and Bergman, L., editors, *2nd Nordic Conference on Advanced Information Systems Engineering.* Springer-Verlag.

Dietz, J. (1992). *Leerboek Informatiekundige Analyse.* Kluwer Bedrijfswetenschappen.

Dignum, F. (1989). *A Language for Modelling Knowledge Bases.* PhD thesis, Free University, Amsterdam.

Dijk, A., van Swede, V., and Visser, J. (1989). *Taalkundige informatiesystemen ontwikkeld met Grammars.* Pandata Uitgeverij, Rijswijk.

Dik, S. (1987). A typology of entities. In van der Auwera and Goossens, editors, *Unknown title.* Unknown publisher.

Dik, S. C. (1989). *The Theory of Functional Grammar, part 1: The Structure of the Clause.* Foris Publications, Dordrecht, Holland.

EDR (1988). Electronic dictionary project. Technical report, Japan Electronic Dictionary Research Institute, Ltd.

El Emam, K., Quintin, S., and Madhavji, N. (1996). User Participation in the Requirements Engineering Process. *Journal of Requirements Engineering*, 1(1).

Evans, R. and Gazdar, G. (1996). DATR: A language for lexical knowledge representation. *Computational Linguistics*, 22(2):167–216.

Fellbaum, C. (1993). English verbs as a semantic net. Technical report, Princeton University.

Fillmore, C. (1968). The case for case. In Bach and Harms, editors, *Universals in Linguistics*, pages 1–88. Holt, Rinehart, and Winston, New York.

Gamut, L. (1991). *Logic, Language, and Meaning*, volume 2, Intensional Logic and Logical Grammar. The University of Chicago Press.

Glass, A. L. and Holyoak, K. J. (1986). *Cognition.* Random House, New York, second edition.

Goguen, J. (1994). Requirements Engineering as the Reconciliation of Social and Technical Issues. In Jirotka, M. and Goguen, J., editors, *Requirements Engineering: Social and Technical Issues*, pages 165–200. Academic Press, London.

Grant, R. and Naylor, D. (1991). *Better Than Life.* Penguin Books.

Gruber, T. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220.

Guiraud, P. (1959). *Problèmes et méthodes de la statistique linguistique.* Reidel, Dordrecht.

Gulla, J. (1993). *Deep Explanation Generation in Conceptual Modelling Environments.* PhD thesis, University of Trondheim.

Gulla, J. and Willumsen, G. (1993). Using Explanations to Improve the Validation of Executable Models. In Rolland, C., Bodart, F., and Cauvet, C., editors, *The Proceedings of the Fifth International Conference on Advanced Information System Engineering (CAiSE'93)*, number 685 in Lecture Notes in Computer Science, pages 118–142.

Gulla, J. A., van der Vos, B., and Thiel, U. (1996). Retrieving Conceptual Models on the Basis of Word Semantics. In van de Riet, R. et al., editors, *Applications of Natural Language to Information Systems*, pages 115–126. IOS Press.

Gupta, P. and Sykes, J. A. (1996). A Basis for Natural Language Analysis of Part-Whole Relationships in Fact-Based Conceptual Modelling. In van de Riet, R. et al., editors, *Applications of Natural Language to Information Systems*, pages 210–221. IOS Press.

Guthrie, L., Pustejovsky, J., Wilks, Y., and Slator, B. M. (1996). The Role of the Lexicon in Natural Language Processing. *Communications of the ACM*, 39:1:63–72.

Habermas, J. (1984). *The Theory of Communicative Action: Reason and Rationalization of Society.* Polity Press, Cambridge.

Halpin, T. (1995). *Conceptual Schema and Relational Database Design.* Prentice Hall Australia, second edition.

Halpin, T. and Harding, J. (1993). Automated support for verbalization of conceptual schemas. In Brinkkemper and Harmsen, editors, *Proceedings of the 4th European Workshop on Next Generation CASE Tools, Paris*, pages 151–161.

Hofmann, H. (1993). Requirements engineering: A survey of methods and tools. Technical report, Institut für Informatik der Universität Zürich.

Hoppenbrouwers, J. (1997). Generative Specification of a Template-Based Textual Analysis Tool. In *Proceedings of the Third International Workshop on Applications of Natural Language to Information Systems, June 26–27, Simon Fraser University, Vancouver, Canada*, pages 191–203.

Hoppenbrouwers, J. et al. (1997a). A CASE Tool Based on Textual Analysis. Submitted to CAISE'97.

Hoppenbrouwers, J., van der Vos, B., and Hoppenbrouwers, S. (1996). NL Structures and Conceptual Modelling: The KISS Case. In van de Riet, R., Burg, J., and van der Vos, A., editors, *Application of Natural Language to Information Systems*, pages 197–209. IOS Press.

Hoppenbrouwers, J., van der Vos, B., and Hoppenbrouwers, S. (1997b). NL Structures and Conceptual Modelling: Grammalizing for KISS. *Data and Knowledge Engineering*, 23(1):79–92.

Ingram, D. (1989). *First Language Acquisition: method, description and explanation.* Cambridge University Press.

Jackson, R., Embley, D., and Woodfield, S. (1995). Developing formal object-oriented requirements specifications: A model, tool, and technique. *Information Systems*, 20(4):273–289.

Jarke, M., Pohl, K., Dömges, R., Jacobs, S., and Nissen, H. (1994). Requirements Information Management: the NATURE approach. *Ingenerie des Systems d'Informations (Special Issue on Requirements Engineering)*, 2(6).

Kaminsky, J. (1969). *Language and Ontology.* Southern Illinois University Press.

Kim, Y.-G. and March, S. T. (1995). Comparing data modeling formalisms. *Communications of the ACM*, 38(6).

Kristen, G. (1994). *Object Orientation: The KISS Method.* Addison-Wesley.

Kyle, J. and Woll, B. (1985). *Sign Language.* Cambridge University Press.

Landis, T., Herrmann, D., and Chaffin, R. (1987). Development differences in the comprehension of semantic relations. *Z. Psychologie*, 195(2):129–139.

Leibniz, G. W. (1903). Elementa characteristica universalis. In Couturat, L., editor, *Opuscules et Fragments inédits de Leibniz*, pages 42–92. Ancienne Librairie Germer Bailliere, Paris.

Lenat, D. (1995a). CYC: a Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 13(11).

Lenat, D. (1995b). Steps to Sharing Knowledge. In Mars, N., editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KB&KS'95)*, pages 1–2. IOS Press, Amsterdam.

Lenat, D. and Guha, R. (1990). *Building Large Knowledge Bases.* Addison-Wesley.

Littlewood, W. (1984). *Foreign and Second Language Learning.* Cambridge University Press.

Loucopoulos, P. and Karakostas, V. (1995). *System Requirements Engineering.* McGraw-Hill Book Company Europe.

Luqi and Goguen, J. (1997). Formal Methods: Promises and Problems. *IEEE Software*, 14(1):73–85.

Mackenzie, L. (1987). The representation of nominal predicates in the fund. Technical Report WPFG no. 25, Free University of Amsterdam.

Mann, W. and Thompson, S. (1987). Rhetorical Structure Theory: Description and Construction of Text Structures. In Kempen, G., editor, *Natural Language Generation: New results in Artificial Intelligence, Psychology, and Linguistics.* Martinus Nijhoff.

McFetridge, P., editor (1997). *Proceedings of the Third International Workshop on Applications of Natural Language to Infomation Systems*, Simon Fraser University, Burnaby, B.C., Canada.

Meersman, R. (1982). The ridl conceptual language. Technical report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels.

Miller, G. A. (1993). Nouns in wordnet: A lexical inheritance system. Technical report, Princeton University.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). Introduction to wordnet: An on-line lexical database. Technical report, Princeton University.

Minsky, M. (1975). A framework for representing knowledge. In Winston, editor, *The Psychology of Computer Vision*, pages 211–280. McGraw-Hill, New York.

Mockapetris, P. (1983). Domain names: Concepts and facilities. Request for Comments 882, Network Working Group.

Moulin, B. and Creasy, P. (1992). Extending the conceptual graph approach for data conceptual modelling. In van de Riet, R. and Meersman, R., editors, *Linguistic Instruments in Knowledge Engineering*. Elsevier Science Publishers BV.

Naur, P. (1960). Report on the Algorithmic Language ALGOL 60. *Communications of the ACM*, 3(5):299–314.

NGGO (1994). *13 Methoden voor object-georiënteerde systeemontwikkeling*. Tutein Nolthenius.

Nijssen, G. M. (1993). *Universele Informatiekunde*. PNA Publishing BV, Beutenaken.

Nijssen, G. M. and Halpin, T. A. (1989). *Conceptual Schema and Relational Database Design*. Prentice Hall, Sydney.

Ogden, C. and Richards, I. (1923). *The Meaning of Meaning*. Harcours, Brace, and World, New York, 8th edition 1946 edition.

OMG (1996). The Common Object Request Broker Architecture: Architecture and Specification. Technical document PTC/96-03-04, Object Management Group.

Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley.

Pollard, C. and Sag, I. (1987). *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Procter, P., editor (1987). *Longman Dictionary of Contemporary English*. Longman.

Pustejovsky, J. (1991). The Generative Lexicon. *Computational Linguistics*, 17(4).

Pustejovsky, J. (1993). Linguistic Constraints on Type Coercion. Technical Report CS-93-171, Brandeis University.

Quirk, R. and Greenbaum, S. (1973). *A University Grammar of English.* Longman.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language.* Longman.

Robinson, M. and Bannon, L. (1991). Questioning representations. In Bannon, L., Robinson, M., and Schmidt, K., editors, *Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, page 219.

Rosch, E. (1977). Classification of real world objects: origins and representation in cognition. In Johnson-Laird, P. and Wason, P., editors, *Thinking: readings in cognitive science.* Cambridge University Press.

Russell, G., Caroll, J., and Warwick, S. (1990). Multiple Default Inheritance in a Unification-Based Lexicon. In Daelemans, W. and Gazdar, G., editors, *Inheritance in Natural Language Processing*, pages 93–101. Institute for Language Technology and Artificial Intelligence, Tilburg University, The Netherlands.

Sagan, Drake, Lomberg, et al. (1978). *Murmurs of Earth.* Planetary Society.

Searle, J. (1969). *Speech acts; an essay in the philosophy of language.* Cambridge University Press.

Sebesta, R. W. (1989). *Concepts of Programming Languages.* Benjamin/Cummings.

Senko, M. (1976). NIAM as a detailed example of the ANSI SPARC architecture. In Nijssen, G., editor, *Modelling in Data Base Management Systems*, pages 73–94. North-Holland.

Sinclair, J., Hanks, P., Fox, G., Moon, R., and Stock, P., editors (1987). *Collins Cobuild English Language Dictionary.* Collins, London and Glasgow.

Sowa, J. F. (1983). *Conceptual Structures, information processing in mind and machine.* Addison-Wesley, Reading, Massachusetts.

Sparck Jones, K. (1964). *Synonymy and Semantic Classification.* Edinburgh Information Technology Series.

Steuten, A. (1997). Business Conversations from a Conversation-Analytical and a Functional Grammar Perspective. In Butler, C., Connolly, J., Gatward, R., and Vismans, R., editors, *Discourse and Pragmatics in Functional Grammar.* Mouton de Gruyter, Berlin.

Steuten, A. and van Reijswoud, V. (1996). The Interpretation of Business Communication: The Application of Functional Grammar and the Transaction Process Model. In Dignum, F., Dietz, J., Verharen, E., and Weigand, H., editors, *Proceedings of the First International Workshop on Communication Modelling, Tilburg, The Netherlands, July 1–2, 1996.* Springer.

Storey, V. C. (1991). Meronymic relationships. *Journal of Database Administration*, 2(3):22–35.

Strachey, C. (1967). Fundamental Concepts in Programming Languages. Lecture Notes for the International Summer School in Computer Programming, Copenhagen.

Taylor, D. (1990). *Object-Oriented Technology: A Manager's Guide.* Servio Corporation.

ter Hofstede, A., Proper, H., and van der Weide, T. (1993). Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523.

Theodoulidis, B. and Alexakis, P. (1993). Deductive Repository Technology: Supporting Effective Domain Modelling and Validation. In *Proceedings of the 4th Workshop on the Next Generation of CASE Tools, Paris.*

Ulijn, J. M. and Strother, J. B. (1995). *Communicating in Business and Technology: From Psycholinguistic Theory To International Practice.* Peter Lang GmbH, Frankfurt.

van de Riet, R. (1989). MOKUM: An Object-Oriented Active Knowledge Base System. *Data and Knowledge Engineering*, 4(1):21–42.

van de Riet, R. and Burg, J. (1996). Modeling Alter Egos in Cyberspace: Who is Responsible? In *World Conference of the Web Society (WebNet '96).* AACE.

van de Riet, R., Burg, J., and van der Vos, A., editors (1996). *Applications of Natural Language to Information Systems.* IOS Press.

van de Riet, R. and Meersman, R., editors (1992). *Linguistic Instruments in Knowledge Engineering.* North-Holland. Proceedings of the 1991 Workshop on Linguistic Instruments in Knowledge Engineering, Tilburg, The Netherlands, January 17–18 1991.

van Griethuysen, J. and Jardine, D. (1984). De infomodbenadering van informatiemodellering. *Informatie*, (6):423–443.

van Griethuysen, J., Jardine, D., and Bryant, J., editors (1981). *Concepts and Terminology for the Conceptual Schema.* International Standards Organization.

van Reijswoud, V. E. (1996). *The Structure of Business Communication.* PhD thesis, Delft University of Technology, The Netherlands.

Verharen, E. M. (1997). *A Language-Action Perspective on the Design of Cooperative Information Agents.* PhD thesis, Tilburg University.

Verheijen, G. and van Bekkum, J. (1982). Niam: An information analysis method. In Verrijn-Stuart, A., Olle, T., and Sol, H., editors, *Proceedings on IFIP TC-8 Conference on Comparative Review of Information Systems Methodologies (CRIS-1)*, pages 537–589. North-Holland.

Vosse, T. (1994). *The Word Connection: Grammar-based Spelling Error Correction in Dutch.* PhD thesis, Rijksuniversiteit Leiden.

Vossen, P. (1995). *Grammatical and Conceptual Individuation in the Lexicon.* PhD thesis, University of Amsterdam.

Weigand, H. (1990). *Linguistically Motivated Principles of Knowledge Base Systems.* Foris Publications, Dordrecht, Holland.

Weigand, H. and Hoppenbrouwers, S. (1997). The Dynamic Lexicon from a Functional Perspective. In Hengeveld, K., Sánchez, J., and Olbertz, H., editors, *Proceedings of the 7th international conference on Functional Grammar, Cordoba, 1996*, volume 1, The Lexicon.

Weigand, H. and Verharen, E. (1996). Interoperable Transactions in Business Models—A Structured Approach. In Constantopoulos, P., Mylopoulos, J., and Vassiliou, Y., editors, *Proceedings of the 8th CAiSE Conference*, pages 193–209.

Weinrich, U. (1964). Webster's Third: A Critique of its Semantics. *International Journal of American Linguistics*, 30:405–409.

Wiederhold, G. (1995). Value-added mediation in large-scale information systems. In *Database Applications' Semantics, IFIP TC-2 DS-6.*

Wieringa, R. (1995). An introduction to requirements traceability. Technical Report Technical Report IR-389, Vrije Universiteit Amsterdam.

Wieringa, R. (1996). *Requirements Engineering: Frameworks for Understanding.* Wiley.

Wieringa, R., Meyer, J.-J., and Weigand, H. (1989). Specifying dynamic and deontic integrity constraints. *Journal of Data and Knowledge Engineering*, 4(2):157–191.

Wijers, G. and Heijers, H. (1990). Automated Support of the Modelling Process: A View based on Experiments with Expert Information Engineers. In Steinholz, B., Solvberg, A., and Bergman, L., editors, *Proceedings of the Second Nordic Conference on Advanced Information Systems Engineering (CAiSE'90)*, number 436 in Lecture Notes in Computer Science.

Wilks, Y. (1996). Natural Language Processing. *Communications of the ACM*, 39(1):60–62.

Winston, M., Chaffin, R., and Herrmann, D. (1987). A taxonomy of part-whole relations. *Cognitive Science*, 11:417–444.

Wintraecken, J. (1985). *Informatie-analyse volgens NIAM in theorie en praktijk.* Academic Service.

Wu, C.-H. and Flynn, D. (1995). NALABA: A Natural Language Based Integrity Requirements Validation System. In *First International Workshop on Applications of Natural Language to Data Bases (NLDB'95), Versailles*, pages 167–182.

Yokoi, T. (1995a). The EDR Electronic Dictionary. *Communications of the ACM*, 38(11).

Yokoi, T. (1995b). The Impact of the EDR Electronic Dictionary on Very Large Knowledge Bases. In Mars, N., editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KB&KS'95)*, pages 13–21. IOS Press, Amsterdam.

Zernik, U., editor (1991). *Lexical acquisition: exploiting online resources to build a lexicon.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Author Index

List of Acronyms

AI Artificial Intelligence

AMAZON AutoMAtische Zins ONtleding

ASCII American Standard Code for Information Interchange

AV Attribute-Value

BNF Backus-Naur Form

BRM Binary Role Model(ing)

CASE Computer Aided Software Engineering

CELEX CEnter for LEXical information

CG Conceptual Graph

CM Conceptual Modeling

CORBA Common Object Request Broker Architecture

CPL Conceptual Programming Language

CS Computer Science

DATR Acronym unknown; a lexical representation language

DBMS Data Base Management System

DEMO Dynamic Essential MOdeling

EBNF Extended Backus-Naur Form

EDI Electronic Data Interchange

EDR Electronic DictionaRy (project)

EoD Environment of Discourse

ER Entity-Relationship (model)

FG Functional Grammar

FORM Formal Object-Role Modeling

FORML Formal Object-Role Modeling Language

GRAMMARS GRAMMAticale Relationele informatieSystemen

HTML HyperText Markup Language

IS Information System

ISO International Standards Organization

IT Information Technology

KISS Kristen's Iteration, Selection, and Sequence (model)

KISS Kristen Information & Software Services BV

LA Lexicon Administrator

LDL Lexicon Definition Language

LDOCE Longman's Dictionary of Contemporary English

LIKE Linguistic Instruments in Knowledge Engineering (project)

LISA-D Language for Information Structure and Access Descriptions

LIX Lexical Information eXchange

LMS Lexicon Management System

LOT Lexical Object Type

LQL Lexicon Query Language

LWB Lexicographers' Work Bench

MERODE Model-driven Entity-Relationship Object-oriented DEvelopment

MRD Machine Readable Dictionary

NATURE Novel Approaches to Theories Underlying Requirements Engineering (project)

NIAM Nijssen's Information Analysis Method

NL Natural Language

NLP Natural Language Processing

NOLOT Non-Lexical Object Type

NORM Natural Object-Role Modeling

NT Not Today

OED Oxford English Dictionary

OI Object-Interaction (model)

OLAP On-Line Analytical Processing

OO Object-Oriented, Object-Orientation

ORM Object-Role Modeling

OT Object Type

PSM Predicator Set Model

RDBMS Relational Data Base Management System

RIDL Reference and IDea Language

SC Subject-Communication (model)

SIKS School for Information and Knowledge Systems

SoA State of Affairs

SQL Structured Query Language

TCP/IP Transmission Control Protocol/Internet Protocol

TREVI Text Retrieval and Enrichment for Vital Information

UI Universele Informatiekunde

UoD Universe of Discourse

WWW World-Wide Web

Index

W

Curriculum Vitae

Jeroen Hoppenbrouwers was born in Bergeyk (Noord-Brabant) on July 22, 1967. He finished secondary school (Gymnasium β) in 1985 at Hertog Jan College in Valkenswaard. After having completed the first year of the Mechanical Engineering program at Eindhoven University of Technology, he continued to study Computer Science and Technical Communication at the same University. In 1991, he graduated on a Master's thesis about textual structures which he wrote in cooperation with Digital Equipment Corporation under supervision of Jan Ulijn and Ron Verheijen. After his graduation, Jeroen worked at Digital's CEC in Amsterdam where he did research on indexing and retrieval systems before he accepted a PhD student position at Tilburg University's Infolab.

Currently, he works as senior researcher for EIT, an econometrics and information technology consultancy organization affiliated to Tilburg University, where he contributes to language/information science projects such as TREVI and the *Grammalizer*. He is interested in large scale document production and processing, NL manipulation, distributed autonomous technology, and thin clients.

Jeroen is not married and lives in Valkenswaard.

Center for Economic Research, Tilburg University, The Netherlands
Dissertation Series

No.	Author	Title
1	P.J.J. Herings	Static and Dynamic Aspects of General Disequilibrium Theory; ISBN 90 5668 001 3
2	Erwin van der Krabben	Urban Dynamics: A Real Estate Perspective. An institutional analysis of the production of the built environment; ISBN 90 5170 390 2[2]
3	Arjan Lejour	Integrating or Desintegrating Welfare States? A qualitative study to the consequences of economic integration on social insurance; ISBN 90 5668 003 x
4	Bas J.M. Werker	Statistical Methods in Financial Econometrics; ISBN 90 5668 002 1
5	Rudy Douven	Policy Coordination and Convergence in the EU; ISBN 90 5668 004 8
6	Arie J.T.M. Weeren	Coordination in Hierarchical Control; ISBN 90 5668 006 4
7	Herbert Hamers	Sequencing and Delivery Situations: a Game Theoretic Approach; ISBN 90 5668 005 6
8	Annemarie ter Veer	Strategic Decision Making in Politics; ISBN 90 5668 007 2
9	Zaifu Yang	Simplicial Fixed Point Algorithms and Applications; ISBN 90 5668 008 0
10	William Verkooijen	Neural Networks in Economic Modelling. An Empirical Study; ISBN 90 5668 010 2
11	Henny Romijn	Acquisition of Technological Capability in Small Firms in Developing Countries; ISBN 90 5668 009 9
12	W.B. van den Hout	The Power-Series Algorithm. A Numerical Approach to Markov Processes; ISBN 90 5668 011 0
13	Paul W.J. de Bijl	Essays in Industrial Organization and Management Strategy; ISBN 90 5668 012 9
14	Martijn van de Ven	Intergenerational Redistribution in Representative Democracies; ISBN 90 5668 013 7

[2]Copies can be ordered from Thesis Publishers, P.O. Box 14791, 1001 LG Amsterdam, The Netherlands. Phone +31 20 6255429, fax +31 20 6203395, e-mail `thesis@thesis.aps.nl`

No.	Author	Title
15	Eline van der Heijden	Altruism, Fairness and Public Pensions: An Investigation of Survey and Experimental Data; ISBN 90 5668 014 5
16	H.M. Webers	Competition in Spatial Location Models; ISBN 90 5668 015 3
17	Jan Bouckaert	Essays in Competition with Product Differentiation and Bargaining in Markets; ISBN 90 5668 016 1
18	Zafar Iqbal	Three-Gap Analysis of Structural Adjustment in Pakistan; ISBN 90 5668 017 x
19	Jimmy Miller	A Treatise on Labour: A Matching-Model Analysis of Labour-Market Programmes; ISBN 90 5668 018 8
20	Edwin van Dam	Graphs with Few Eigenvalues. An interplay between combinatorics and algebra; ISBN 90 5668 019 6
21	Henk Oosterhout	Takeover Barriers: the good, the bad, and the ugly; ISBN 90 5668 020 x
22	Jan Lemmen	Financial Integration in the European Union: Measurement and Determination; ISBN 90 5668 021 8
23	Chris van Raalte	Market Formation and Market Selection; ISBN 90 5668 022 6
24	Bas van Aarle	Essays on Monetary and Fiscal Policy Interaction: Applications to EMU and Eastern Europe; ISBN 90 5668 023 4
25	Francis Y. Kumah	Common Stochastic Trends and Policy Shocks in the Open Economy: Empirical Essays in International Finance and Monetary Policy; ISBN 90 5668 024 2
26	Erik Canton	Economic Growth and Business Cycles; ISBN 90 5668 025 0
27	Jeroen Hoppenbrouwers	Conceptual Modeling and the Lexicon; ISBN 90 5668 027 7

Stellingen

behorend bij het proefschrift

Conceptual Modeling and the Lexicon

Jeroen Hoppenbrouwers

1. Een taalkundig fundament onder een conceptueel model maakt het definiëren van semantische functies, entiteiten en relaties minder arbitrair, en dus wetenschappelijker.[1]

2. Het idee dat er slechts één 'beste' of 'juiste' manier is om met een hedendaagse methode een domein te modelleren getuigt van een arrogant gebrek aan respect voor de inzichten van anderen en een overschatting van de semantische dekking van het gebruikte paradigma.

3. Het modelleren van een domein op basis van generieke taal levert een minder gedetailleerd maar wel duurzamer model op dan het modelleren op basis van gegevensstructuren of andere door logica en/of techniek geïnspireerde gronden.

4. Standaard NIAM en KISS gebruiken weliswaar allebei natuurlijke taal om tot hun modellen te komen, maar stijgen niet wezenlijk uit boven de syntaxis van de taal; de werkelijke semantische inzichten dienen door een analist op eigen initiatief te worden toegevoegd. Dit leidt tot arbitraire analyses die vaak ontwerpbeslissingen bevatten. Introductie van een Lexicon kan een groot deel van deze willekeur wegnemen.

5. Grootschalige gedistribueerde informatiesystemen kunnen op den duur niet zonder een algemeen taalgebaseerd Lexicon als hulpmiddel bij de interne onderhandelingen, omdat éénduidige, gestandaardiseerde en deterministische communicatieprotocollen zoals EDIFACT en CORBA IDL niet de semantische expressiemogelijkheden en flexibiliteit bieden die van dergelijke systemen zullen worden geëist.

6. Als een wetenschapper geen dubbel werk verricht, verdient hij het niet om in het Engels *r*esearcher te worden genoemd.

7. Het is de moeite waard te overwegen om informatica-onderzoek voortaan niet meer bij de faculteiten Wiskunde maar bij de faculteiten Letteren onder te brengen.

---

[1] Weigand, *Linguistically Motivated Principles of Knowledge Base Systems*, Foris, 1989.

JEROEN HOPPENBR

communication at Eindhoven University of Technology. He carried out his Ph.D research at the Infolab (Faculty of Economics, Tilburg University). Currently he is senior researcher at EIT, an econometrics and information technology consultancy organization related to Tilburg University.

'Conceptual Modeling and the Lexicon' investigates the linguistic aspects of conceptual modeling, concentrating on the terminology part. The author combines theoretical ideas and empirical facts from various scientific fields, such as cognitive psychology, computer science, lexicography, psycholinguistics, software engineering, philosophy, and linguistics, to develop a central Lexicon which should improve mutual communication between all the people who work on a new information system. A preliminary design of a Lexicon Management System and some actual implementations complete the work.

ISBN 90-5668-027-7

CentER Dissertation